國立交通大學

電子工程學系 電子研究所

博 士 論 文

針對 FIR 與 FFT 演算法於超大型積體電路
實作上之解析式面積最佳化技術

Analytical Area Optimization for VLSI Implementations of FIR and FFT Algorithms

研 究 生：林步青

指導教授：周景揚 博士

　　　　　黃俊達 博士

中 華 民 國 一 〇 三 年 六 月

針對 FIR 與 FFT 演算法於超大型積體電路實作上之解析式面積最佳化技術

Analytical Area Optimization for VLSI Implementations of FIR and FFT Algorithms

研究生：林步青　　Student: Bu-Ching Lin

指導教授：周景揚 博士　　Advisors: Jing-Yang Jou

黃俊達 博士　　Juinn-Dar Huang

國立交通大學

電子工程學系　電子研究所

博士論文

A Dessertation
Submitted to Department of Electronics Engineering & Institute of Electronics
College of Electrical & Computer Engineering
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy
in
Electronics Engineering & Institute of Electronics

June 2014
Hsinchu, Taiwan, Republic of China

中華民國一〇三年六月

針對 FIR 與 FFT 演算法於超大型積體電路實作上之解析式面積最佳化技術

研究生：林步青　　　　指導教授：周景揚博士 黃俊達博士

國立交通大學

電子工程學系及電子研究所 博士班

摘要

在過去幾十年中，隨著通信系統的複雜度急劇增加，數位訊號處理演算法被廣泛地採用，例如有限脈衝響應(FIR)濾波器和快速傅利葉轉換(FFT)。其中，多常數乘法(MCM) 是在處理輸入資料與常數的乘法時，使用一組加法器取代常規乘法器，其概念更是廣泛地被應用在有限脈衝響應濾波器的設計中。在過去，雖然已有很多降低加法器用量之演算法被提出以達到面績縮小的目的，但是，它們並未考慮每個加法器的實際位元數，而這將會導致估計的硬體成本不夠精確。因此這篇論文中，我們提出了一個保證位元數的多個常數乘法最佳化演算法，著重於最大限度地減少加法器的總位元數，而不是僅考慮減少加法器總數。首先，構建基於給定係數的子表達式圖表，繼而導出一組針對最小化加法器位元數之條件，最後使用整數線性規畫得到最佳化的結果。實驗結果顯示，該演算法的確可以有效地減少所需的加法器位元數並且優於所有的現行技

術。

此外，快速傅利葉轉換處理器在眾多以數位訊號處理為基礎的系統中是一個核心的元件；例如，現代無線通信中的正交分頻多工(OFDM)。許多關鍵的設計參數，如架構，位元長度，和數字格式，都必須非常仔細地考慮。在過去的幾十年，針對不同的設計目標，已經有很多最佳化的管線式快速傅利葉轉換架構被提出。雖然固定的管線式快速傅利葉轉換架構能在合理的硬體成本下提供不錯的處理能力，但是，在針對需要大量處理能力的應用上，它可能仍然無法滿足效能的需求。因此在這篇論文中，我們提出了一種可擴展的多路徑延遲累積器式之快速傅利葉轉換架構及其相應的硬體設計產生器，在給定的處理能力條件下，它能夠迅速地產生對應的快速傅利葉轉換核心。實驗結果顯示，此方法所產生之快速傅利葉轉換器比現有的可折疊式多路徑延遲累積器式快速傅利葉轉換架構，面積更小且功率效率更高。

除此之外，我們亦提出了一個快速傅利葉轉換器最佳化的設計流程。在固定位元長度的條件下，正確的調整每一個蝴蝶級的定點表示之數值，以最大化輸出級的信號量化雜訊比(SQNR)。所提出的流程採用機率分佈模型來模擬每個階段的輸出信號的機率行為。由於量化和飽和運算所導致的雜訊可以靜態分析，以了解在進行縮放決策時的影響。因此，不需耗費時間的模擬分析，我們所提出的方法即可有效地決定每一個蝴蝶級的最適當的數字格式，從而最佳化整個輸出級的信號量化雜訊比。此外，建議的流程能夠處理各種快速傅利葉轉換點數、快速傅利葉轉換演算法、字元長度、以及輸入信號的機率分佈。實驗結果顯示，我們的方法可以在 8192 點且以 2 為基數的快速傅利葉轉器處理器中節省 3 位元的字元長度，而且不會對輸出級的信號量化雜訊比造成影響。使用我提出的靜態尺規最佳化的技術所創建的快速傅利葉轉器處理器的信號量化雜訊比可以近似於一個配有額外的動態尺規化方法，但不需要其額外龐大的硬體成本。

Analytical Area Optimization for VLSI Implementations of FFT and FIR Algorithms

Student: Bu-Ching Lin

Advisor: Dr. Jing-Yang Jou
Dr. Juinn-Dar Huang

Department of Electronics Engineering
Institute of Electronics
National Chiao Tung University

Abstract

As the complexity of communication systems has grown dramatically in the past decades, digital signal processing algorithms are extensively used, such as finite-impulse-response filter (FIR) and fast Fourier transform (FFT). Meanwhile, the notion of multiple constant multiplication (MCM) is extensively adopted in FIR designs. A set of adders are used to replace regular multipliers for the multiplications between input data and constant filter coefficients. Though many algorithms have been proposed to reduce the total number of adders in an MCM block for area minimization, they do not consider the actual bitwidth of each adder, which may not estimate the hardware cost well enough. Therefore, we propose a bitwidth-aware MCM optimization algorithm that focuses on minimizing the total number of adder bits rather than the adder count. It first builds a subexpression graph based on the given coefficients, derives a set of constraints for adder bitwidth minimization, and then optimally solves the problem through integer linear programming (ILP). Experimental results show that the proposed algorithm can effectively reduce the required adder bit count and outperforms the existing state-of-the-art techniques.

The FFT processor serves as one of core components in numerous DSP-based systems, such as OFDM in modern wireless communication. The key design parameters, such as architecture, wordlength, and number format, must be all considered very carefully. Many pipelined FFT architectures optimized for different objectives have been proposed in past few decades. Though a fixed pipelined FFT architecture can generally provide good throughput at reasonable hardware cost, it may still fail to meet the performance demand for throughput-hungry design cases. In this dissertation, we propose an expandable MDC-based FFT architecture as well as the corresponding hardware design generator, which is capable of automatically producing an FFT core under a given throughput constraint. The experimental results show that the proposed methodology can generate smaller and power-efficient implementations than the existing foldable MDC-based FFT architecture.

Besides, in this dissertation, we also propose an optimization flow that properly scales fixed-point numeric values at each butterfly stage to maximize the output SQNR under a fixed wordlength constraint. The proposed flow utilizes probability distribution to model the probabilistic behavior of the output signal at each stage. The computation errors due to quantization and saturation operations are statically analyzed before making scaling decisions. Therefore, without a need of time-consuming simulation, our method can efficiently determine the most appropriate number format for each stage and thus optimize the overall output SQNR. Besides, the proposed flow is capable of handling various FFT sizes, FFT algorithms, wordlengths, and input signal distributions. Experimental results indicate that the wordlength can be reduced about three bits for an 8K-point radix-2 memory-based

FFT processor without compromise in the output SQNR. Furthermore, the FFT processor created using our static scaling optimization technique can produce a comparable output quality as the one equipped with an extra dynamic number scaling unit, which requires significantly more hardware logic.

Acknowledgment

First and foremost, I would like to express my greatest appreciation to my advisors, Professor Jing-Yang Jou (周景揚) and Professor Juinn-Dar Huang (黃俊達) for their suggestions, guidance, and patience. Their wisdom, knowledge, and commitment to the highest standards inspired and motivated me.

Also, I would like to thank the involved members in the projects, Jhih-Hung Lu (呂智宏), Yu-Hsiang Wang (王毓翔), Ming-En Stone (石銘恩), Yao-Chung Hsu (許耀中), and Yi-Fang Chen (陳奕方). Without the seamless cooperation, the projects would not be so successful. Thanks to Geeng-Wei Lee (李耿維), Cheng-Yeh Wang (王成業), Zwei-Mei Lee (李瑞梅), Chia-I Chen (陳嘉怡), Ya-Shih Huang (黃雅詩), and Wan-Yu Lee (李婉毓) for several useful discussions. A special thanks to all EDA members for the wonderful time we share together.

I would like express my sincere appreciation to Yi-Fang Huang (黃儀芳), my wife, without whom this effort would have been worth nothing. Your love support, and constant patience have taught me so much about sacrifice, discipline, and compromise. I would appreciate my family for their patient wait and my father education philosophy "no wonder sometimes incompetent people win out in imperial examinations, but does these mean that the erudite ones will be neglected?"

BU-CHING LIN

National Chiao-Tung University

June, 2014

Content

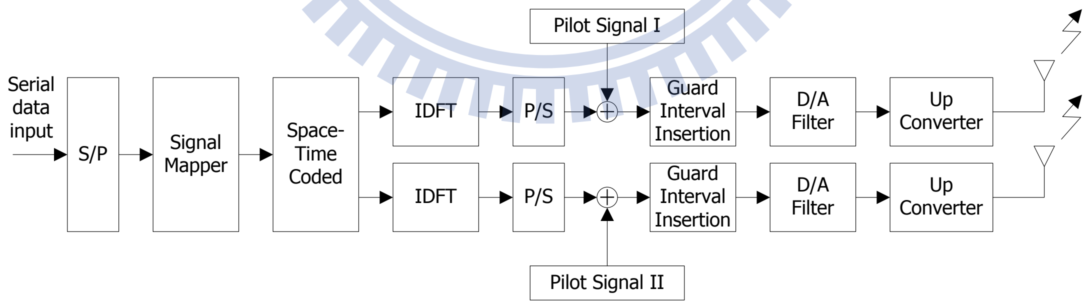
List of Figures

List of Tables

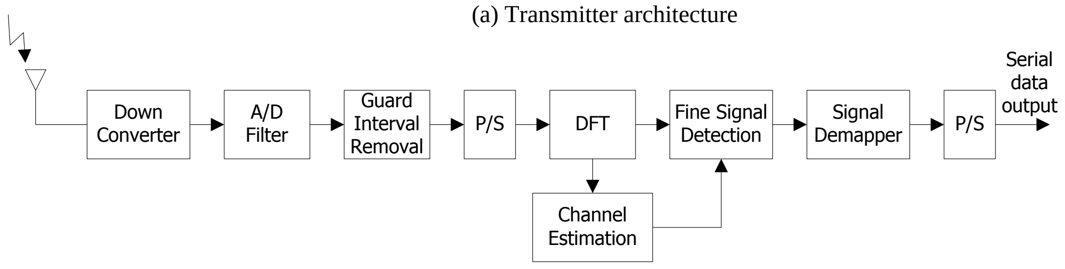
Chapter 1

Introduction

Digital signal processing (DSP) algorithms have become the foundation of today's audio/video and communication systems. The well-known DSP algorithms, including finite-impulse-response (FIR) filters and fast Fourier transform (FFT), have attracted the attentions of a number of researchers over several decades. For example, orthogonal frequency-division multiplexing (OFDM) is known for the advantages of tolerance to inter-frequency interference and robustness against frequency-selective fading. It is widely used in modern broadband wireless communication systems, such as digital audio/video broadcast (DAB/DVB) [1], local area network (WLAN, IEEE 802.11a/g/n) [2], and networks for multiple users (IEEE 802.16, WiMAX, and LTE) [3-5]. In OFDM systems, one of the most computationally intensive operations is



(a) Transmitter architecture



(b) Receiver architecture

Figure 1. An example pilot-based STBC/OFDM communication system.

modulation/demodulation, which requires the filters to maximize the signal-to-noise ratio (SNR) and a dedicated FFT processor for discrete Fourier transform (DFT) computation [5-7]. Figure 1 illustrates an example of a pilot-based STBC/OFDM transmitter and receiver communication architecture [8]. Since the filter and FFT are required for processing each stream data, they are obviously the major computation blocks in a high-data-rate baseband system. Also note that, the area of the FFT core is reported about 25% of that of an IEEE 802.11a baseband processor [9, 10]. Thus, the DSP algorithms, such as the filter and FFT cores, must be carefully designed to minimize the hardware cost.

1.1 MCM-Based FIR Designs

For most of DSP algorithms, multiplication is one of the most frequently-used essential operations. For example, each input of an N-tap FIR filter should be multiplied by N constant coefficients. The multiplication between an input data value and a set of constants is also referred to as the multiple constant multiplication (MCM). Since the multiplier is not a small functional unit in hardware implementation, its usage should be reduced whenever possible. Meanwhile, a constant multiplication can actually be accomplished through a set of adders along with proper bit shifting rather than a real multiplier. For instance, a constant multiplication that always multiplies the input value x by 5 (i.e., $y = 5 \times x$) can be computed as $y = (x << 2) + x$. That is, a regular multiplier can be safely replaced with just an adder. Note that shifting a value by a fixed number of bits can be simply achieved via signal wiring in hardware implementation, which is virtually at no cost. As compared with the trivial implementation using a regular multiplier, it is obvious that the one with adders can usually reduce the hardware cost. Moreover, the area saving is likely to be more significant if the multiplication results with multiple constants are required simultaneously. For example, two constant multiplications, $y_1 = 5 \times x$ and $y_2 = 13 \times x$, can be accomplished by only two adders: $y_1 = (x << 2) + x$ and $y_2 = (x << 3) + y_1$. Consequently, the notion of multiplier-less MCM has been widely adopted for implementing area-efficient digital filters.

For an adder, the bitwidth is directly proportional to its area, delay, as well as power. For example, Figure 2 illustrates the area, latency, and power consumption of a ripple carry adder under the TSMC 0.18μm technology. The area, power consumption, and latency of an 8-bit adder are 116 gates, 0.55 mW, and 5.9 ns, respectively, which are approximately half of those of a 16-bit adder (225 gates, 1.05 mW, and 10.2 ns). Therefore, it is crucial to take the adder bitwidth into account very seriously while implementing a multiplier-less MCM block since it is solely made of adders.

The problem of multiplier-less constant multiplication is actually equal to the classical addition chain problem, in which a constant multiplication is replaced by a series of additions [11]. However, it is much more complicated to effectively share the outputs of intermediate additions among multiple constant multiplications. Existing

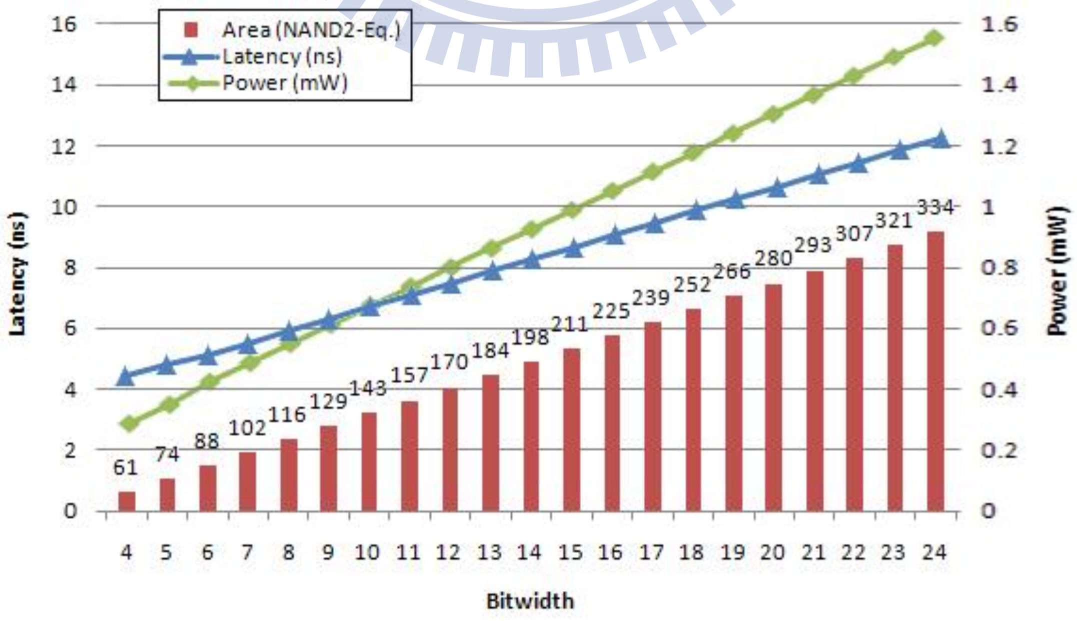


Figure 2. Area, latency, and power scale with the adder bitwidth.

multiplier-less MCM algorithms for adder minimization can be divided into two major categories: 1) the graph-based algorithms [12-19], and 2) the digit-based algorithms [20-31]. Given a set of constants, a graph-based algorithm gradually builds the corresponding graph representation in a bottom-up fashion and uses some heuristic method to explore the chances for subexpression sharing among those constants. In general, they can provide a solution with fairly good quality in a short runtime; however, optimality is not guaranteed. In contrast, a digit-based algorithm generates a set of number decompositions for subexpression sharing based on a specific numeric representation. The ILP solver is sometimes utilized to produce an adder-minimal solution [23, 27-30] at the cost of longer runtime [32, 33].

Note that most of the previous studies regard the adder count minimization as the optimization goal and neglects the fact that every adder has its own area and delay cost due to different bitwidth. Hence, in this dissertation, we propose a new algorithm that minimizes the overall adder bitwidth instead of the total adder count. Meanwhile, the methods presented in the previous works [28, 29] use a pre-computed lookup table to store all feasible number decompositions of all constants within a fixed precision (e.g., 13 bits) no matter they are required or not. In contrast, our algorithm dynamically creates a subexpression graph that merely contains necessary information. The experimental results demonstrate that our algorithm can achieve a reduction on the total adder bit count by up to 10% for some test cases as compared with the existing state-of-the-art techniques. The details of the proposed ILP-based bitwidth-aware subexpression sharing algorithm are elaborated in Chapter 2.

1.2 FFT Architectures and Designs

An FFT core has several important design parameters, such as *base architecture*, *size*, and *wordlength*. Since the FFT algorithm was first proposed in

[34], various architectures have been developed and optimized for different goals. The architectures of FFT processors can be roughly divided into two major categories: memory-based and pipeline-based. Memory-based architectures usually consist of a butterfly unit and certain number of memory blocks for providing low-cost designs. However, it is very difficult for them to achieve real-time processing at low clock frequency. Alternatively, a pipelined architecture consists of multiple stages to provide higher throughput at the cost of more hardware. In general, memory-based architectures are suitable for FFT processors where the hardware cost is an issue and the FFT size is not smaller than 512 [3]. Pipeline-based architectures are typically feasible for applications with smaller FFT sizes. Meanwhile, the proper FFT size varies for different applications. For example, the size can be 128, 256, 512, 1024, or 2048 for WiMAX applications; and 256, 512, 1024, or 2048 for DAB systems. Hence, for a specific application, the requested FFT core should be well configured to meet its own unique requirements.

Many pipelined FFT architectures [35-44] have been proposed in past few decades. For a specific architecture, one way to accelerate its computation is to increase the clock rate. For example, the conventional Radix-2 Multi-path Delay Commutator (R2MDC) takes 1023 clock cycles for a 1024-point FFT computation. Assume that a butterfly operation takes 20 ns in a given technology; and as a result the execution time of a complete 1024-point FFT computation requires 20.5 us. If the system requires even faster response time (< 20.5 us), a higher operation frequency is demanded. The other way to provide higher throughput is to expand the datapath of the original architecture. However, expanding a pipelined FFT architecture is not always trivial because of internal dissimilar data permutation patterns and complicated controlling logic. The difficulty generally increases as the degree of parallelism rises. Thus, a generator that can automatically provide those expanded architecture variants is demanded for saving design costs drastically. To deal with this issue, in this dissertation we propose an Expandable Multi-path Delay Commutator (EMDC) based FFT architecture and the corresponding generator targeting high-performance applications. Due to the page limitation, we only demonstrate the details of the proposed Radix-2^2EMDC architecture in Chapter 3. However, the proposed approach can be easily applied to other MDC-based architectures, such as Radix-2^3MDC and Radix-2^4MDC.

Furthermore, the scaling optimization on FFT designs with a fixed wordlength at each stage needs to be considered; that is, the output wordlength of every stage is the same as its input wordlength. Memory-based designs naturally fulfill this requirement, while a considerable part of pipeline-based designs also choose to meet the same requirement since the fixed wordlength is still preferred due to hardware cost and critical-path delay considerations.

While crafting a practical FFT hardware design, the output precision in terms of signal-to-quantization-noise (SQNR) ratio is regarded as a key design requirement. In practice, FFT algorithms are commonly implemented using fixed-point arithmetic instead of floating-point arithmetic for hardware cost reduction. That is, only a limited number of bits are available to represent a signal or coefficient value. As a result, rounding and truncation operations inevitably introduce noises, which are referred to as quantization noises. Besides, addition and subtraction operations may also cause overflow errors (noises) during computations. Although extending the wordlength can relieve the accuracy loss, the hardware cost and the critical-path delay are increased accordingly.

Therefore, several number scaling methods, either static or dynamic, have been proposed to improve the output SQNR [1, 45-64]. Oppenheim et al. [49]

proposed a simple static scaling method that always increases the integer part by one bit for each radix-2 stage to prevent overflows. However, this method suffers significant quantization errors if the wordlength is fixed. In addition, methods based on dynamic scaling have also been proposed for the SQNR improvement. Block Floating Point (BFP)-based methods employ intermediate buffers to store and analyze a block of output values, and then dynamically determine an appropriate number format for that block to achieve a better SQNR [1, 45, 46, 51, 52]. However, all of these dynamic methods suffer a notable increase in area, power, and latency as well as need a more complicated control unit to achieve a similar quality of result (QoR). Consequently, most FFT designers rely on static instead of dynamic scaling optimization techniques to determine a proper number format for each stage [50].

Previous static scaling techniques can be roughly classified into three major categories: simulation-based approaches [58, 59], analytical approaches [47, 49, 50, 60-68], and a hybrid of previous two [48, 69, 70]. The simulation-based approaches try to find a good number format through lengthy iterations. In contrast, the analytical methods can determine a good number format very efficiently through a static numeric analysis without invoking time-consuming simulation. However, the analysis results are generally too pessimistic and lead to a larger wordlength than required. Therefore, the hybrid approaches are proposed to determine the number format and shorten the simulation time simultaneously. Meanwhile, the works mentioned above [47-50, 58-70] all assume they can arbitrarily determine the wordlength at each stage. However, in memory-based FFT designs, the wordlength (the width of the memory block) is always fixed. To the best of our knowledge, the problem of static scaling under a fixed wordlength constraint has not been well addressed yet.

In this dissertation, we propose a scaling optimization technique based on the static probability analysis that can rapidly determine the best number format at each butterfly stage under a fixed wordlength constraint. Given a probability distribution of input signals, a selected FFT algorithm, and a wordlength constraint, the proposed technique can maximize the overall output SQNR through the static number format analysis and optimization stage by stage. Compared to previous works, our method offers the following three contributions: 1) providing a probability model that can abstract the behavior of fixed-point arithmetic logic; 2) preventing the use of time-consuming and pattern-dependent simulation throughout the entire optimization process; 3) minimizing the required wordlength in a hardware implementation under a given SQNR target without demanding extra hardware components and complicating control logic compared to other existing static approaches [49, 50].

1.3 Dissertation Organization

The rest of this dissertation is organized as follows. In Chapter 2, an ILP-based bitwidth-aware subexpression sharing for area minimization in filter design is presented. In Chapter 3, we demonstrate an expandable MDC-based FFT architecture and its generator targeting for the high-performance applications. Then, a probability-based static scaling optimization flow for fixed wordlength FFT processors is developed in Chapter 4. Finally, the concluding remarks and the future works are given in Chapter 5.

Chapter 2

Bitwidth-Aware Subexpression Sharing in FIR Filter

2.1 FIR Filter Implementation

In this section, we first introduce the fundamentals of an FIR filter design. Then, three different number representations as well as the related algorithms are briefly reviewed here.

2.1.1 Fundamentals of FIR filter design

A general form of a linear time-invariant *N*-tap FIR filter can be expressed as a convolution involving the last *N* input data and *N* constant filter coefficients. The output *y*(*n*) can be calculated as:

$$y(n) = \sum_{k=0}^{N-1} c_k \times x(n-k) \qquad (1)$$

where

1. $x(n)$ is the input sequence,
2. $y(n)$ is the corresponding output sequence,
3. c_k are constant filter coefficients, and
4. N is the filter length.

Figure 3 illustrates a general fully-parallel transposed architecture of FIR filter, which requires *N*–1 additions and *N* multiplications to produce a single output value. It is also observed that one of the two inputs of every multiplication is always from the

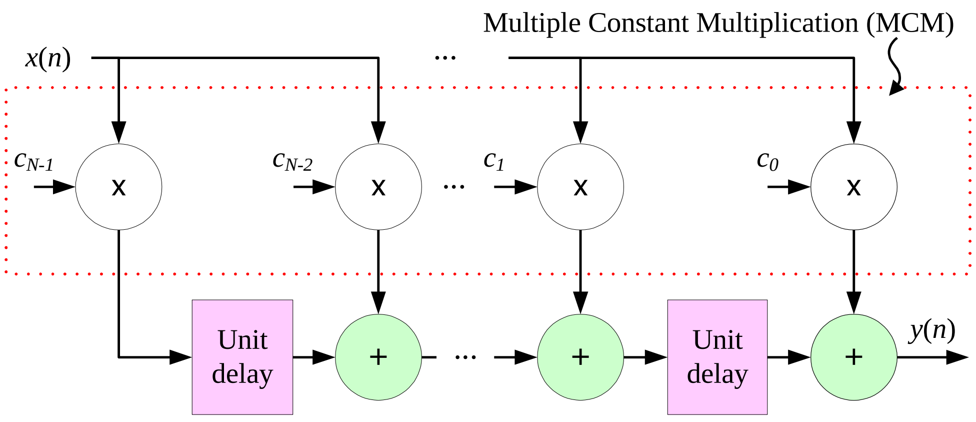


Figure 3 A general architecture of FIR filter.

present input sample. Therefore, an MCM block can be used to produce a set of products between an input value and a given set of constant coefficients. Since all coefficients are fixed-point constants, those constant multiplications can be solely achieved through a series of additions, and thus the use of costly regular multipliers can be completely eliminated as previously mentioned

2.1.2 Number representations

Three different number formats can be used to represent fixed-point constants: pure binary form, the canonical signed digit (CSD) form, and the minimal signed digit (MSD) form. The pure binary form is the trivial unsigned binary representation, where every digit is either 0 or 1. In the CSD representation, three symbols, 0, 1, and $\bar{1}$, are used, where $\bar{1}$ denotes –1. The CSD representation has the following two properties: 1) the count of nonzero digits (i.e., $\bar{1}$ and 1) is minimal; and 2) any two adjacent digits cannot both be nonzero. In addition, the CSD representation for a

Table 1 Three different representations for the number 23

	Representation
pure binary	010111
CSD	$10\bar{1}00\bar{1}$
MSD	$01100\bar{1}$ or $10\bar{1}00\bar{1}$

number is unique, and this is how “canonical” comes from. Similar to the CSD form, the MSD form also adopts the same three symbols and requires that the count of nonzero digits is minimal. Unlike the CSD from, the MSD form allows adjacent nonzero digits, which makes itself no longer a canonical representation. In other words, a number may have multiple valid MSD representations. Note that the CSD representation is also a valid MSD representation. Table 1 gives an example where the number 23 is presented in those three representations with the length of six digits. The number 23 has a unique representation in pure binary form and CSD form, but has two feasible representations in MSD form. Besides, it is a guarantee that for any number the count of nonzero digits in pure binary representation is no less than that in CSD and MSD representations.

In hardware implementation, the count of nonzero digits of a value *c* basically determines the number of required additions to realize the multiplication by *c*. Figure 4 demonstrates three different ways for implementing a constant multiplication by 23. Figure 4(a) shows a direct implementation based on the pure binary representation (i.e., 10111). Since there are four nonzero digits, three adders are required to complete the multiplication (i.e., 23x=16x+4x+2x+x). However, there are only three nonzero

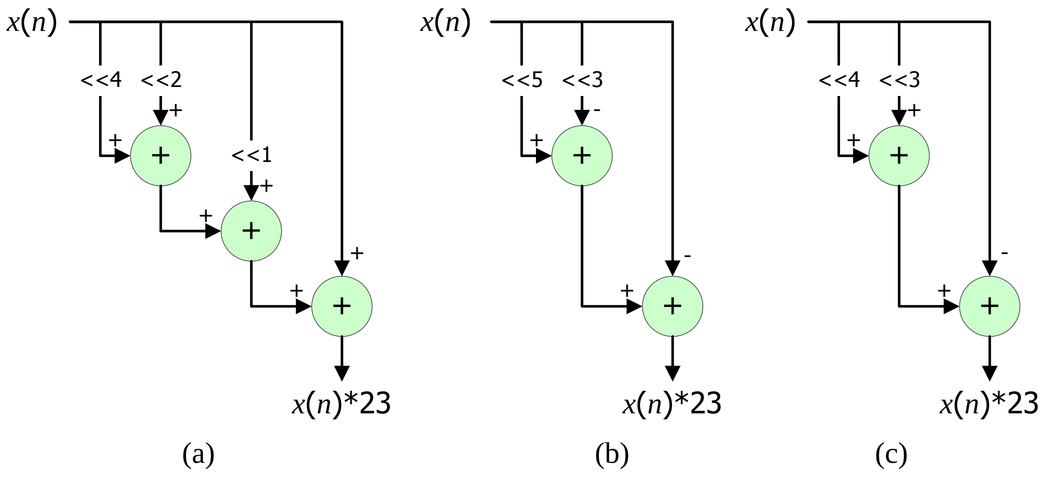


Figure 4 Three different implementations for the number 23.

digits if the CSD format is considered (i.e., $10\bar{1}00\bar{1}$). As shown in Figure 4(b), merely two adders are enough to accomplish the same multiplication (i.e., $23x=32x–8x–x$, a subtraction $a – b$ is regarded as an addition $a + (–b)$ in this dissertation). Since the number 23 has two valid MSD representations, Figure 4(c) illustrates the other one (i.e., $01100\bar{1}$), which also needs two adders only (i.e., $23x=16x+8x–x$). Consequently, CSD and MSD representations are usually adopted in constant multiplications because both of them guarantee the count of nonzero digits for any given constant is always minimal.

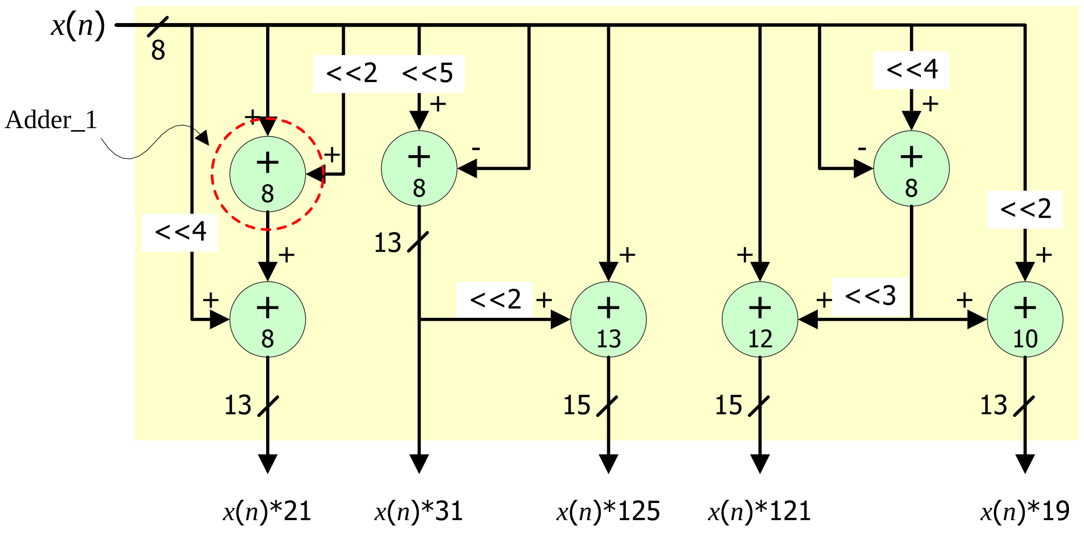
2.1.3 Digit-based algorithms

A number of algorithms have been proposed to decompose a set of constants based on a specified number representation [20-33]. However, most of these previous methods only focus on the minimization of the total adder count. Since the area and delay of an adder is highly dependent on the bitwidth as mentioned, it is unwise to neglect its impact during optimization. For example, two different ways can be used to implement the constant multiplication of $11x$: $(1x+2x)+8x$ or $(1x+8x)+2x$. They both require two adders to complete the multiplication. However, the bitwidth of the result of $(1x+8x)$, is apparently longer than that of $(1x+2x)$. Since a wider result potentially requires wider adders for succeeding additions, it is actually a good idea to take the resultant bitwidth of addition outcome into account for better optimization outcomes.

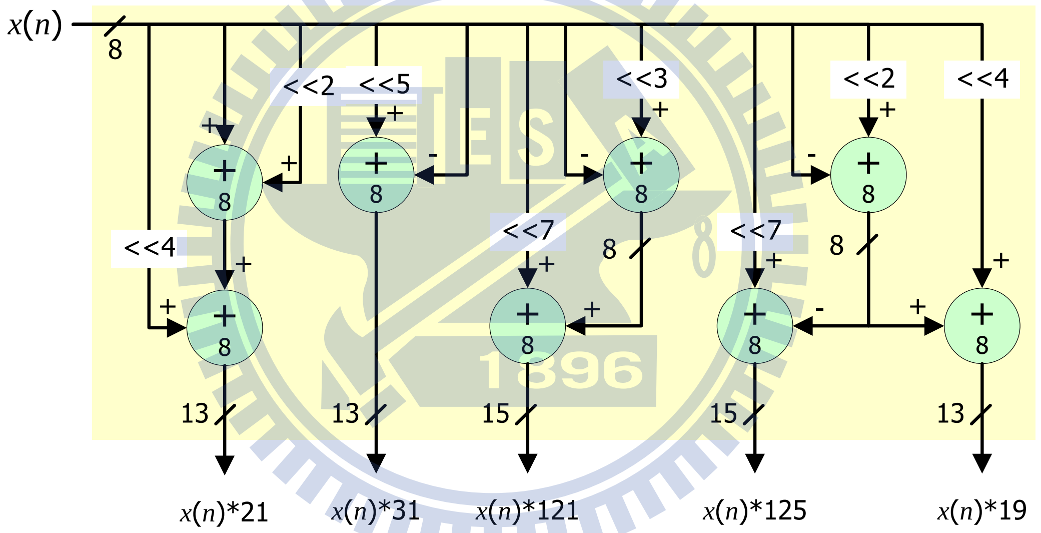
2.1.4 Motivation of our work

Figure 5 (a) illustrates a sample multiplier-less MCM design, where the input x is 8-bit wide. Instead of using five costly multipliers, the five output values (the input x times 19, 21, 31, 121, and 125) can be produced by only seven adders. Note that every adder must be wide enough to ensure the absence of overflow. Typically, an adder with n–1 bits is used to generate an n-bit sum. For example, Adder_1 shown in Figure 5(a) is used to produce an output of $5x = x + (x << 2)$, where the output is 11-bit wide and the trivial implementation of Adder_1 should be 10-bit wide, as depicted in Figure 6 (a). However, it is observed that the rightmost two adder bits in Figure 6(a) are actually redundant due to the 2-bit right shifting (i.e., $x << 2$). As a consequence, Adder_1 can be safely downsized to an 8-bit adder, as shown in Figure 6(b). This example clearly indicates that a more compact adder implementation can possibly be achieved if the relation between two input operands is carefully investigated.

Let us reexamine the implementation shown in Figure 5(a). The number labeled within a circle specifies the minimal bitwidth of the corresponding adder. Hence, a total of 67 bits is required for those 7 adders. Meanwhile, Figure 5(b) illustrates another implementation for the same MCM, which requires 8 adders but only 64 adder bits. That is, the implementation depicted in Figure 5(b) consumes more adders but less adder bits than that shown in Figure 5(a). As previously explained, we consider the solution given in Figure 5(b) is better. However, those previous approaches trying to minimize the adder count would prefer the solution shown in Figure 5(a). Consequently, it motivates us to develop a new area minimization algorithm for MCM designs that concentrates on total adder bitwidth minimization. The details are elaborated in the following two sections.

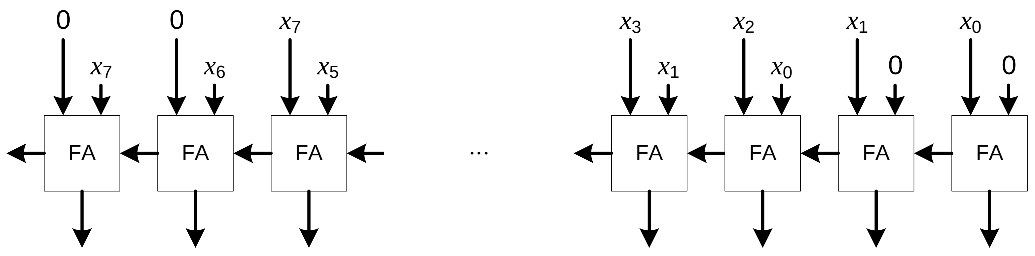


(a) an MCM design with 7 adders and 67 bits

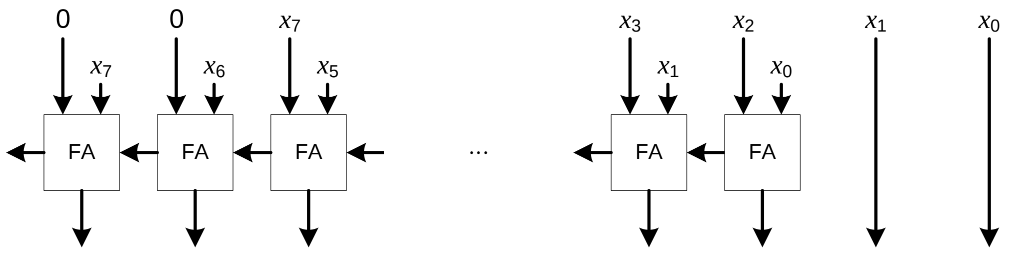


(b) same MCM design with 8 adders and 64 bits

Figure 5 A motivational example of multiplier-less MCM.



(a) a 10-bit adder



(b) an 8-bit adder

Figure 6 Two alternative implementations of Adder_1, $x + (x << 2)$.

2.2 Proposed Algorithm

In this section, we present a bitwidth-aware ILP-based area minimization algorithm for MCM designs. It uses a graph-based approach that keeps track of common subexpressions among constants as well as calculates the exact required bitwidth of each adder simultaneously. The details are described in the following sections.

2.2.1 Number decomposition and bitwidth calculation

The fundamental of the MCM optimization problem is to maximize the common subexpression sharing among given constants. Hence, it is generally true that the optimization outcome could be better if more ways are considered for decomposing a constant, i.e., a larger solution space. However, the number of possible ways for decomposing a number is actually infinite. For example, the number 3 can be achieved as 1+2, 4–1, 5–2, 6–3, 7–4, and so on. Fortunately, not all of them are appropriate while constructing an area-efficient solution. As explained in Section 2.2, the number of adders (A_z) needed to accomplish a constant multiplication by z is equal to the number of nonzero bits of z (B_z) minus one, i.e., $A_z = B_z - 1$. Hence, there is no reason to decompose z as $x + y$ if $B_z < B_x + B_y$ during optimization. For instance, it is not wise to decompose the number 3 as 5–2. In our method, a set of target constants $D = \{d_i\}$ is first converted into $C = \{ c_i \mid d_i = c_i \times 2^l, c_i \text{ is an odd number}\}$. That is, all constants c_i are assumed odd numbers initially. Next, for every constant c with k (where $k > 1$) nonzero digits in its MSD representation, the proposed algorithm merely considers a finite set of number decompositions complying with the following format:

$$c = d + e = d + f \times 2^{l}, l > 0 \tag{2}$$

where d and f must be odd, as well as e must be even.

For example, the set of decompositions for the number 153 ($1010\bar{1}001$ in MSD) is enumerated in Table 2. Actually, a number decomposition $c = d + e$ is created by partitioning the nonzero digits in c's MSD form into two nonempty disjoint groups. The group contains the least significant digit (LSD) actually defines the value of d (odd), while the other group specifies the value of f (even). As shown in the last column of Table 2, the nonzero digits are partitioned into gray and non-gray groups. Therefore, for a number c that has m valid MSD representations and every one includes k nonzero digits, the total number of possible decompositions of c can be formulated as:

$$\begin{aligned} &\frac{1}{2} \times m \times \left(\binom{k}{1} + \binom{k}{2} + \cdots + \binom{k}{k-1} \right) = \\ &\frac{1}{2} \times m \times \left(\left(\sum_{i=0}^{k} \binom{k}{i} \right) - 2 \right) = m \times \left(2^{k-1} - 1\right) \end{aligned} \tag{3}$$

For instance, the total number of decompositions for 153 is $1 \times (2^{4-1} - 1) = 7$ according to (3).

For a decomposition $c = d + f \times 2^{l}$, both $|d|$ and $|f|$ are called the subexpressions of c. For example, the number 153 has eleven different subexpressions – {1, 3, 5, 7, 15, 19, 25, 33, 121, 129, 161}. Hence, according to (2), the decompositions and the subexpressions of any odd number larger than 1 can be identified using the approach described above.

An adder is required to carry out a decomposition of a constant multiplication $cx = dx + fx \times 2^{l} = p + (q << l)$. Assume p is m-bit long and q is n-bit long, the adder bitwidth can thus be determined based on the following two scenarios:

Table 2 All feasible decompositions of the number 153

index	$d + e$	$d + f \times 2^l$	153 in MSD
1	$\boxed{1} + 152$	$\boxed{1} + 19 \times 2^3$	$10101\bar{1}00\boxed{1}$
2	$161 + \boxed{(-8)}$	$161 + \boxed{(-1) \times 2^3}$	$1010\boxed{\bar{1}}001$
3	$121 + \boxed{32}$	$121 + \boxed{1 \times 2^5}$	$10\boxed{1}0\bar{1}001$
4	$25 + \boxed{128}$	$25 + \boxed{1 \times 2^7}$	$\boxed{1}010\bar{1}001$
5	$\boxed{(-7)} + 160$	$\boxed{(-7)} + 5 \times 2^5$	$1010\boxed{\bar{1}}00\boxed{1}$
6	$\boxed{33} + 120$	$\boxed{33} + 15 \times 2^3$	$10\boxed{1}0\bar{1}00\boxed{1}$
7	$\boxed{129} + 24$	$\boxed{129} + 3 \times 2^3$	$\boxed{1}010\bar{1}00\boxed{1}$

1) $m \geq n + l$: the adder bitwidth is $m - l$,

2) $m < n + l$: the adder bitwidth is n.

The bitwidth of the addition result is resolved by the larger value between m and $n + l$, namely, $max(m, n + l)$. However, since each of the least l bits generates no carry during addition in both scenarios, the required adder bitwidth can be safely reduced to $max(m, n + l) - l$, as illustrated in Figure 7. In general, an adder with smaller bitwidth occupies smaller area and has shorter delay, as aforementioned.

While revisiting the previous decomposition example of 153, we further assume the input x is 8-bit wide, which is used to produce the output 153x. By Table 2, a possible decomposition is $153 = 121 + 32 = 121 + 1 \times 2^5$, which indicates that the output 153x can be obtained from the summation of 121x and 32x. Obviously, 32x is only 5-bit left shift of x. Also note that the left shift requires no additional hardware in the real implementation. Meanwhile, 121x needs to be further decomposed based on (2) in the same manner and its bitwidth is 15 bits (i.e., $8 + \lceil \log_2 121 \rceil$). The adder type required to sum up 121x and 32x is the one illustrated in Table 7(a), where $m = 15$, $n =$

8, and $l = 5$. Therefore, the required bitwidth of the adder is 15 – 5 = 10 bits.

On the other hand, 153 can also be decomposed in the form of 153 = 25 + 128 = 25 + 1 $\times$ 2^7, which implies 153*x* comes from adding 25*x* from 128*x*. Similarly, 128*x* is merely 7-bit left shift of *x* and 25*x* is 13-bit wide (i.e., 8 + $\lceil \log_2 25 \rceil$). Alternatively, the adder type required to sum up 25*x* and 128*x* is the one shown in Figure 7(b), where $m = 13$, $n = 8$, and $l = 7$. Consequently, the required bitwidth of the adder is 8 bits.

2.2.2 Bitwidth-aware multiplier-less MCM design flow

Figure 8 gives the overall flow of the proposed bitwidth-aware ILP-based area minimization algorithm for MCM designs. At first, all constants (odd numbers) are collected into the constant-for-decomposition set *C*; that is, every constant in *C* needs to be further decomposed. Besides, the resultant subexpression set *S* is created to keep track of all constants that may be utilized during the MCM block construction. Then,

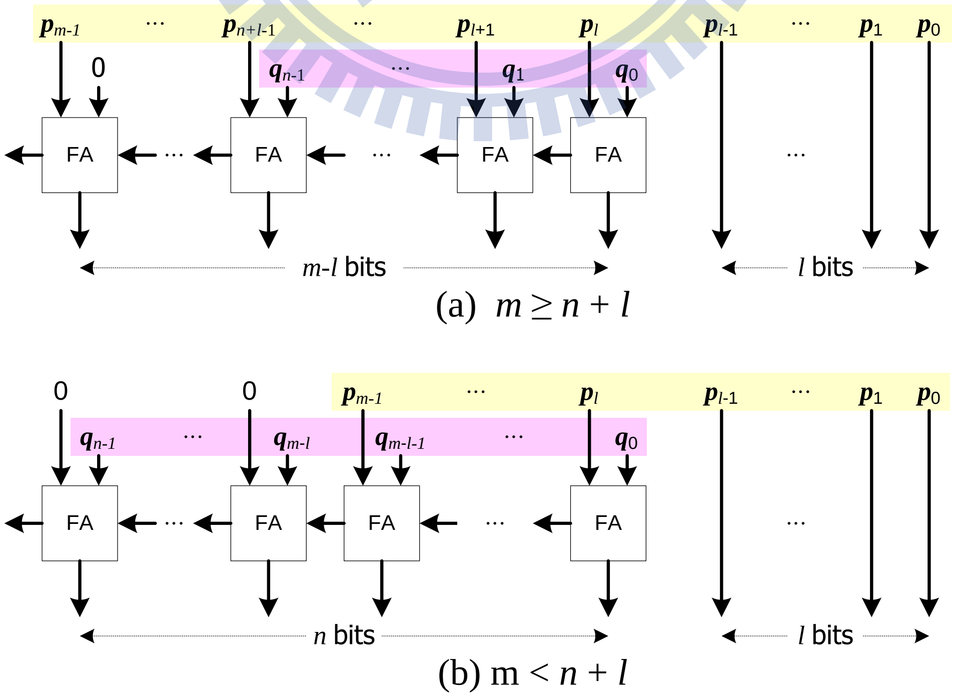


Figure 7 Adder bitwidth calculation.

an arbitrary constant *c* is removed from *C* for decomposition. Based on the specified number representations, i.e., pure binary, CSD, or MSD, all decompositions of *c* are enumerated and the associated hardware cost in terms of total adder bit count is calculated using the method presented in Section 3.1.

Next, *c* is added into *S* after all decompositions of *c* are identified. For every subexpression of *c* that has not been present in *S* yet, it is added into *D* for further decomposition. This process is not terminated until *D* is empty. As a result, the set *S* contains all possible constant numbers that may appear in the final MCM block.

While performing constant number decomposition, the proposed approach concurrently builds a subexpression graph to keep track of all feasible decompositions for a given constant *c*. The graph also records the cost of every decomposition (i.e., adder bit count). Finally, based on this subexpression graph, a set of corresponding ILP constraints can be derived and then an ILP solver is utilized to produce an MCM design solution with the minimal total adder bits. The details of the ILP formulations are given in Section 4 later. Conventional look-up table based approaches require a

```
1.  $C \leftarrow$ { all constant numbers }
2.  $S \leftarrow$ { 1 }
3.  while ( $C$ is not empty )
4.      Remove an arbitrary constant $c$ from $C$
5.      Add $c$ into $S$
6.      foreach ( decomposition $d$ of $c$ )
7.          Identify two subexpressions $s$ & $t$ of $d$
8.          Calculate the required adder bit count
9.          Record $d$ in the subexpression graph $G$
10.         Add $s$ & $t$ into $C$ if they are not in $S$ yet
11. Derive ILP constraints from $G$
12. Find a solution with minimal adder bit count by ILP
```

Figure 8 The proposed algorithm flow.

pre-computed table to store all decompositions of every odd number within a fixed bitwidth (e.g., 13-bit). Therefore, the table can become very huge as the bitwidth increases. In contrast, the proposed algorithm merely enumerates decompositions for a limited number of subexpressions.

2.3 Example of Subexpression Graph Construction

In this subsection, the CSD representation is in use for simplicity. Note that the proposed algorithm is applicable to the MSD one. We also use an example to demonstrate how a subexpression graph is dynamically constructed. In the following example, the 8-bit input is multiplied by two constant numbers, 21 and 153. First, these two constants are transformed in CSD form.

$$21_{10} = 10101_2 = 10101_{CSD}$$

and

$$153_{10} = 10011001_2 = 1010\bar{1}001_{CSD}$$

According to its CSD form, each constant can be further decomposed as a set of subexpressions based on the method described in Section 3.1

$$21 = 1 + 5 \times 2^2 = 17 + 1 \times 2^2 = 5 + 1 \times 2^4$$

and

$$\begin{aligned}153 &= 1 + 19 \times 2^3 = 161 - 1 \times 2^3 = 121 + 1 \times 2^5 = 25 + 1 \times 2^7 \\ &= -7 + 5 \times 2^5 = 33 + 15 \times 2^3 = 129 + 3 \times 2^3\end{aligned}$$

There are three and seven decompositions for 21 and 153, respectively, which is the same as (3) specifies. After decomposition, we also find that the number 21 has three subexpressions of {1, 5, 17} and the number 153 has eleven subexpressions of {1, 3, 5, 7, 15, 19, 25, 33, 121, 129, 161}. Every subexpression (except 1) needs to be further decomposed for finding out all its feasible decompositions and the associated adder bit count, as the algorithm flow presented in Figure 8.

There are two types of vertices within a subexpression graph *G*: 1) a number vertex is associated with a constant value, and 2) an adder vertex specifies a decomposition and associates the decomposed number with its two subexpressions. When a constant number *c* is removed from *C*, a corresponding number vertex is added into *G* if it is not present in *G*. For every feasible decomposition of *c*, a corresponding adder vertex is added into *G*. Similarly, a number vertex associated with each of two subexpressions, *s* and *t*, is also inserted into *G* if it is not present in *G*. The adder vertex not only relates *c* to both *s* and *t* but also keeps track of the required adder bit count for this decomposition. Also note that every unique number *c* has exactly one corresponding vertex in *G*.

Figure 9 illustrates the partial subexpression graph for 21 and 153. The rightmost adder vertex indicates itself a decomposition of 21 (i.e., a fanin of the number vertex associated with 21). It also shows that 1 and 17 are two subexpressions (i.e., its two fanin number vertices) of this decomposition. In addition, it also specifies that the required adder bitwidth is 11 for this decomposition. There are 3 and 7 various

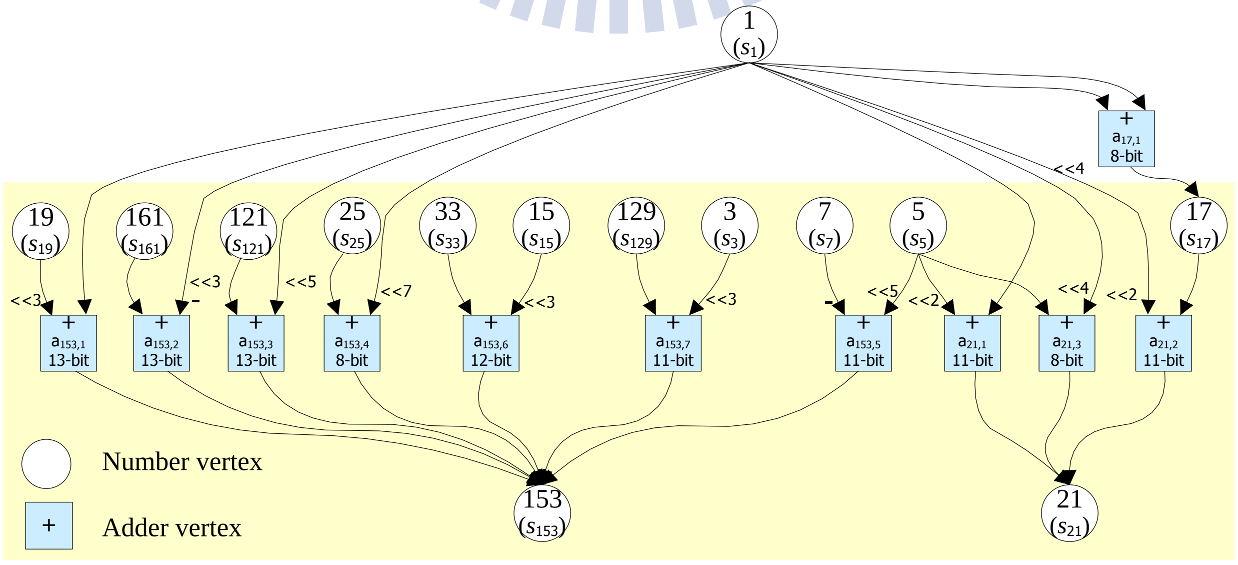


Figure 9 Partial subexpression graph for the numbers 21 and 153.

decompositions for 21 and 153 respectively as shown in Figure 9. A subexpression larger than one should be further decomposed. Except for 17, Figure 9 does not show the succeeding decompositions due to page limitation. Note that Figure 9 also shows that 5 is a common subexpression of both 21 and 153, which implies the hardware cost may be reduced if 5 is shared by both of them in the final implementation. All chances of common subexpression sharing can be exhaustively identified in the proposed subexpression graph.

2.4 ILP Formulation

The problem of bitwidth-aware area minimization for MCM design is thus solved through integer linear programming (ILP). Three types of constraints are derived: 1) an existence constraint guarantees at least one of decompositions is selected for a specified number, 2) a dependency constraint ensures the two corresponding subexpressions would also be implemented if a specified decomposition is selected, and 3) an output constraint guarantees all the output constants of the given MCM are implemented. The ILP-related notations used in this section are given below.

- s_n: a 0-1 variable indicating if the subexpression of the value n is implemented.
- d_n: the count of feasible decompositions of the number n.
- $a_{n,i}$: a 0-1 variable indicating whether the i-th decomposition of the number n is selected for implementation.
- $b_{n,i}$: the required adder bit count for implementing the i-th decomposition of the number n.

2.4.1 Existence constraints

To realize a constant multiplication by $n \geq 1$, at least one of its decomposition must be selected for implementation, which can be formulated as the following constraint:

$$s_n = \max_{1 \le i \le d_n} \{a_{n,i}\} \tag{4}$$

For example, there are three different decompositions for the number 21, as shown in Figure 9. According to (4), at least one of those three decompositions must be selected. That is,

$$s_{21} = \max_{1 \le i \le 3} \{a_{21,i}\} = \max\{a_{21,1}, a_{21,2}, a_{21,3}\}$$

2.4.2 Dependency constraints

As explained in Section 2.3, a decomposition actually implies an adder and its two subexpressions serve as the inputs to the adder. Hence, there is no way to get the addition outcome if those two inputs are not available. That is, if the *i*-th decomposition of the number *n* is selected for realization, both of its two corresponding subexpressions, *x* and *y*, must also be carried out. The constraint can then be formulated as:

$$a_{n,i} = \min\{s_x, s_y\} \tag{5}$$

For instance, the number 153 can be regarded as $33 + 15 \times 2^3$ by the sixth decomposition of 153 shown in Figure 9. Hence, the corresponding dependency constraint can be given as:

$$a_{153,6} = \min\{s_{33}, s_{15}\}$$

2.4.3 Output constraint

Assume that the set C includes all constant numbers of the target MCM block; the following output constraint is applied to ensure that every element $n \in C$ is properly implemented:

$$s_n = 1, \forall n \in C \tag{6}$$

2.4.4 Optimization goal

As aforementioned, the hardware cost of the target MCM block can be lowered if the total adder bit count in use can be minimized. Since every implemented adder must be associated with a variable $a_{n,i}$ set to 1 and the bitwidth of that adder is $b_{n,i}$, the goal of total adder bitwidth minimization can thus be accomplished through setting the objective of the ILP formulation as:

$$\text{minimize} \sum_{\forall n \in C} \sum_{1 \le i \le d_n} a_{n,i} \cdot b_{n,i} \tag{7}$$

subject to (4), (5), and (6).

2.5 Experimental Results

To evaluate the proposed algorithm, we compare it against a widely used graph-based technique revealed in [14] as well as a state-of-the-art digit-based technique presented in [29]. All experiments have been conducted on a Linux platform with two Intel Xeon 2.4 GHz processors and 12 GB main memory. For the preparation of test cases, the Remez algorithm [51, 52] is utilized to randomly generate FIR filters of various types, including low-pass, high-pass, band-pass, and band-stop. Besides, the MSD representation is adopted for the number decomposition, and the Gurobi Optimizer [53] is used as ILP solver.

In Table 3, 12 8-tap filters are evaluated with 12-bit coefficients and input data. For each method, **#adders** reports the total number of adders required in the MCM block, while **#bits** gives the total number of adder bits. Since RAG-n [14] allows the use of right shifters and accepts a mapping result that induces extra adders for a coefficient to maximize global subexpression sharing, it is capable of finding a design solution that is not presented in the solution space for digit-based algorithms. Hence, RAG-n is likely to better minimize the number of adders for an MCM design, as the results show. However, our method can better minimize the number of adder bits for every test case. Even in the case of **bs8-3**, the result of the proposed method needs two more adders but still requires one fewer adder bit than that of RAG-n.

To evaluate the area cost of a real implementation more precisely, we also generate the corresponding Verilog RTL code for a set of 128-tap filter designers and

Table 3 8-tap filter cost comparisons among the three methods

Filters	RAG-n [14]		Ho et al. [28]		Ours	
	#adders	#bits	#adders	#bits	#adders	#bits
lp8-1	6	97	6	91	6	91
lp8-2	8	155	8	129	8	126
lp8-3	7	136	8	127	8	121
hp8-1	6	104	6	104	6	100
hp8-2	8	139	8	134	8	134
hp8-3	7	118	8	120	8	112
bp8-1	10	206	11	190	11	181
bp8-2	9	169	9	161	9	152
bp8-3	10	189	10	180	10	161
bs8-1	9	157	9	162	9	156
bs8-2	7	118	7	109	7	105
bs8-3	8	143	9	152	10	142

Table 4 Synthesis results and comparisons for 12-bit 128-tap FIR filters

	Ho et al. [28]			Ours			Comparisons		
	#adders	#bits	#gates	#adders	#bits	#gates	#adders increased	bit saving (%)	gate saving (%)
lp128-1	68	1,245	23,254	69	1,122	21,487	1	9.9	7.6
lp128-2	57	1,063	20,006	58	987	18,869	1	7.1	5.7
lp128-3	68	1,286	24,443	70	1,175	21,597	2	8.6	11.6
hp128-1	60	1,103	21,305	60	1,025	19,290	0	7.1	9.5
hp128-2	67	1,243	22,004	67	1,165	20,792	0	6.3	5.5
hp128-3	68	1,225	22,264	69	1,122	20,762	1	8.4	6.7
bp128-1	61	1,109	19,952	62	1,004	18,373	1	9.5	7.9
bp128-2	63	1,122	21,469	65	1,048	19,259	2	6.6	10.3
bp128-3	58	1,073	20,090	58	986	18,345	0	8.1	8.7
bs128-1	56	1,026	18,639	57	959	17,856	1	6.5	4.2
bs128-2	61	1,102	20,527	62	1,056	19,215	1	4.2	6.4
bs128-3	56	1,035	19,701	57	977	17,741	1	5.6	9.9

then synthesize the RTL code into the gate-level design based on TSMC 0.18μm technology. In Table 4 and Table 5, two configurations with different coefficient widths, 12-bit and 16-bit, are examined for each filter. In addition, the input is also assumed as wide as the coefficient. Similarly, **#adders** reports the total number of adders in the MCM block, **#bits** gives the total number of adder bits, and **#gates** reveals NAND2-equavilent gate count in the synthesized circuit. Table 4 clearly shows that for every test case the proposed algorithm needs more or same number of adders than the method in [28].

However, our method requires fewer adder bit count in every test case. For 12 12-bit test cases in Table 4, the average reductions on the adder bit count and the gate count are 7.38% and 7.91%, respectively; for 12 16-bit test cases in Table 5, the corresponding reductions are 7.65% and 7.27%, respectively. Note that the

Table 5 Synthesis results and comparisons for 16-bit 128-tap FIR filters

	Ho et al. [28]			Ours			Comparisons		
	#adders	#bits	#gates	#adders	#bits	#gates	#adders increased	bit saving (%)	gate saving (%)
lp128-1	99	2,401	46,789	103	2,195	43,858	4	8.6	6.3
lp128-2	96	2,267	45,783	97	2,129	45,404	1	6.1	0.8
lp128-3	96	2,303	44,290	98	2,205	44,154	2	4.3	0.3
hp128-1	95	2,331	49,248	100	2,133	43,400	5	8.5	11.9
hp128-2	96	2,350	50,826	97	2,151	45,139	1	8.5	11.2
hp128-3	98	2,361	49,281	101	2,170	44,662	3	8.1	9.4
bp128-1	91	2,157	40,536	92	1,996	38,147	1	7.5	5.9
bp128-2	95	2,324	48,145	99	2,094	42,598	4	9.9	11.5
bp128-3	92	2,235	46,108	96	2,067	42,869	4	7.5	7.0
bs128-1	87	2,157	45,151	88	1,975	42,579	1	8.4	5.7
bs128-2	93	2,274	46,591	98	2,104	43,656	5	7.5	6.3
bs128-3	93	2,215	46,102	94	2,061	41,755	1	7.0	9.4

improvements basically remain unchanged as the width of constant number increases from 12-bit to 16-bit, which is a good trend for the future applications. The experimental results also verify that the total adder bit count can better estimate the area cost in a real hardware implementation than the total adder count only, as we claimed previously.

Besides, the proposed method can also adopt the maximum logic depth constraint described in [28] to guarantee the worst-case delay through adding extra depth-related constraints into the ILP formulations. Two different logic depth constraints are examined for comparisons using a set of 12-bit 32-tap FIR filters and the results are shown in Table 5. The average bit count reduction is similar to the previous experiment, which is 5.77% and 7.18% as the maximum logic depth is set to 4 and 5, respectively.

Table 6 Logic depth comparisons for 12-bit 32-tap FIR filters

Filters	Depth = 3				Depth = 4			
	Ho et al. [28]		Ours		Ho et al. [28]		Ours	
	#adders	#bits	#adders	#bits	#adders	#bits	#adders	#bits
lp32-1	16	282	16	271	16	296	16	271
lp32-2	21	363	21	323	21	354	21	323
lp32-3	15	259	15	241	15	250	15	241
hp32-1	22	405	22	373	22	400	22	373
hp32-2	17	294	17	284	17	300	17	284
hp32-3	19	339	19	318	19	334	19	318
bp32-1	25	421	26	382	24	388	24	378
bp32-2	22	399	23	393	21	384	22	373
bp32-3	23	395	23	366	23	417	23	366
bs32-1	23	405	23	346	23	393	23	346
bs32-2	24	419	24	368	23	381	24	368
bs32-3	21	360	21	328	21	356	21	328

In addition, all the ILP problems (including 12-bit and 16-bit 128-tap filters) presented in the experiments can be successfully resolved in less than an hour, which should be considered acceptable. As a consequence, it is conclusive that our proposed algorithm can produce better area-minimized MCM design solutions than the best prior art [29] within an acceptable runtime.

2.5 Summary

In this dissertation, we present an ILP-based bitwidth-aware area minimization algorithm for MCM designs. We first point out that the total adder bit count rather than the total adder count can better estimate the hardware cost in a real implementation. Then, for a given MCM design, those target constants are first represented in a specified number format (MSD in use in this dissertation). Next, a subexpression graph is created to record all feasible decompositions for every target constant. The graph also keeps track of the required adder bitwidth as well as two subexpressions for every decomposition. At last, the area minimization problem is formulated as a set of ILP constraints derived from the subexpression graph and optimally resolved within an acceptable runtime. The experimental results demonstrate that our proposed algorithm can achieve an average reduction of more than 7% on both of the adder bit count and the real gate count. Therefore, we are confident that the proposed approach can outperform the existing state-of-the-art techniques and should be regarded as a better alternative for area minimization in MCM designs

Chapter 3

Expandable MDC-Based FFT

3.1 Overview of the Pipelined FFT Architecture

Pipelined architectures can be divided into two major categories according to the datapath structure. One is Single-path Delay Feedback (SDF) based architecture and the other is Multi-path Delay Commutator (MDC) based architecture. SDF-based architectures use properly-sized local delay feedback loops to correctly schedule the input data for butterfly units. It typically has the advantages of higher hardware utilization rate and less hardware cost. On the contrary, MDC-based architectures first separate the input sequence into two parallel data streams by properly-controlled switches/FIFOs and then direct them into the correct butterfly units. As a result, MDC-based architectures generally demand a bit more hardware resources and larger memory bandwidth but provide higher throughput in return. Nevertheless, though a fixed MDC-based architecture can generally provide good throughput at reasonable hardware cost, it may still fail to meet the target performance requirement for some throughput-hungry design cases.

In [54] and [55], a foldable structure is proposed to provide various design tradeoffs between area and throughput based on the base (Pease) architecture. Since the Pease architecture possesses high regularity, it is extremely easy to fold butterfly units in its implementation either horizontally or vertically. Figure 10 illustrates the fully-expanded 16-point foldable Pease FFT implementation. It is apparent that 4 butterfly columns are identical and thus can be easily 2- or 4-folded horizontally.

Similarly, identical 8 butterfly rows can be 2/4/8-folded vertically as well. With this folding technique, an area-optimized architecture can be tailored to meet the given throughput constraint. However, this customized architecture still requires more area than conventional pipelined architectures when delivering same throughput (shown later). Furthermore, a matrix factorization of FFT computation is also developed in [44] and [55]. Each element in the factored matrix can be expressed as a specific hardware component so that area/performance evaluation can be easily done at the architecture exploration stage. However, [44] and [55] do not consider MDC-based pipelined architectures.

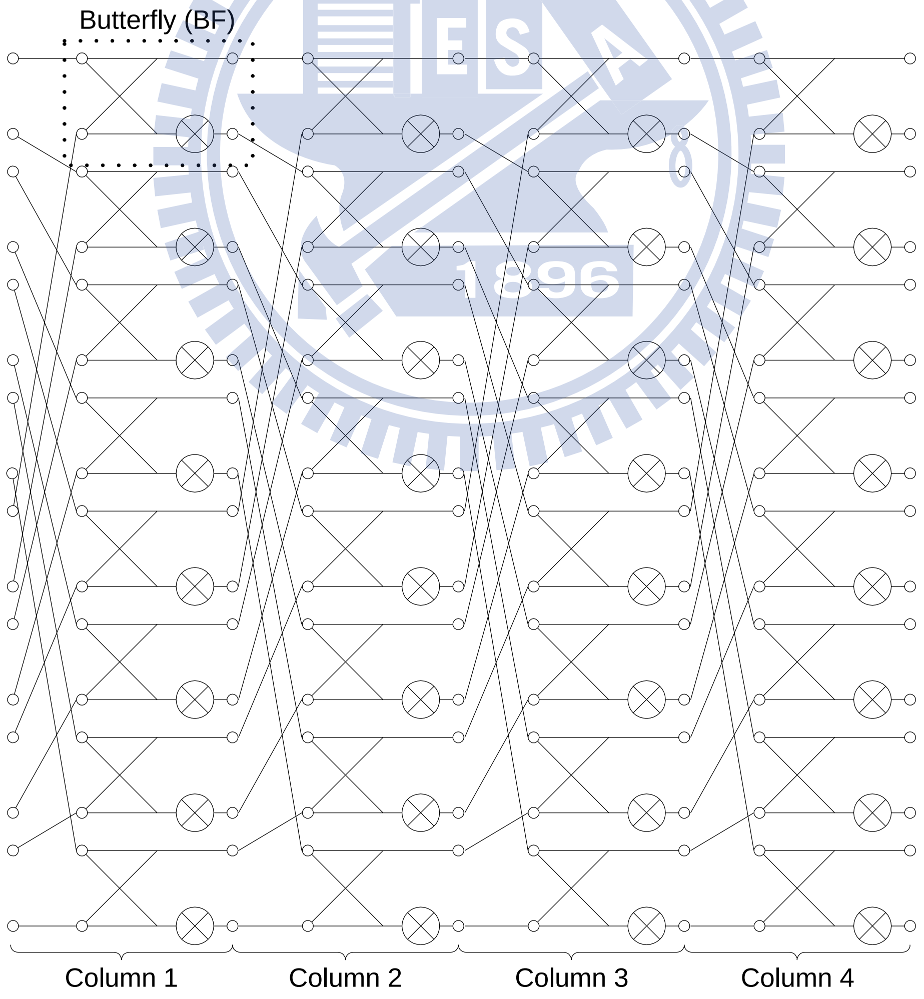


Figure 10 A foldable Pease architecture for 16-point FFT

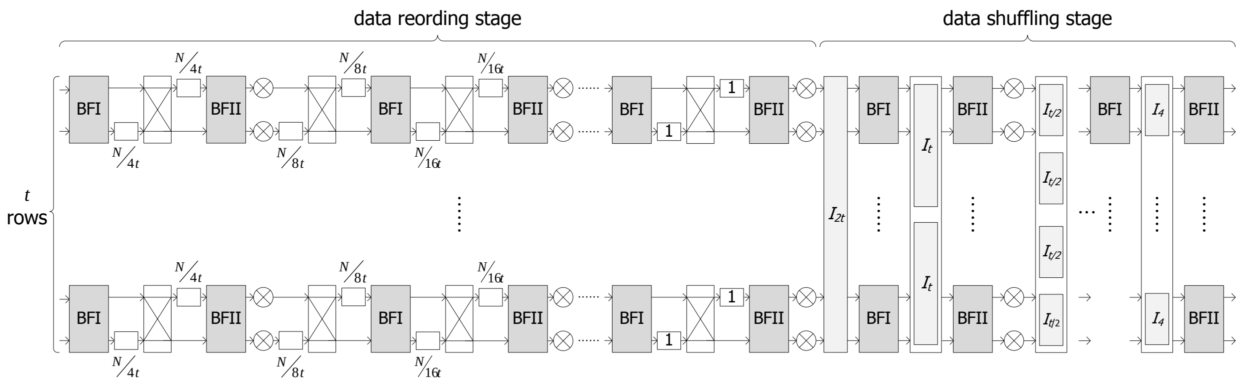


Figure 11 The generic template of R2^{2}EMDC architecture

3.2 Proposed Architecture

In this dissertation, we propose an area-efficient high-throughput **Expandable MDC** (**EMDC**) based FFT architecture. It can be easily applied to conventional MDC-based FFT architectures (such as R2^{2}MDC, R2^{3}MDC, and R2^{4}MDC). Here, we only demonstrate the Radix-2^2 Expandable MDC (R2^{2}EMDC) architecture.

3.2.1 The proposed R2^{2}EMDC architecture

The generic template of the proposed R2^{2}EMDC architecture is presented in Figure 11 Three key parameters are described as follows:

N: the FFT size, where $N = 2^m$, and m is a positive integer.

t: the degree of parallelism obtained from expansion, where $t = 1, 2, 2^2, \ldots, 2^{m-1}$.

I_n: the $n \times n$ interconnection permutation matrix (IPM), where $n = 2^2, 2^3, \ldots, 2^m$.

The proposed architecture is composed of two stages – in addition to butterfly units, the front stage, named data reordering stage, employs FIFOs with specific size and properly-controlled switches to align the data in correct order; while the back stage, named data shuffling stage, deploys a set of precisely-organized IPMs to shuffle the data among different rows to their correct positions (i.e., bit reversing). Note that

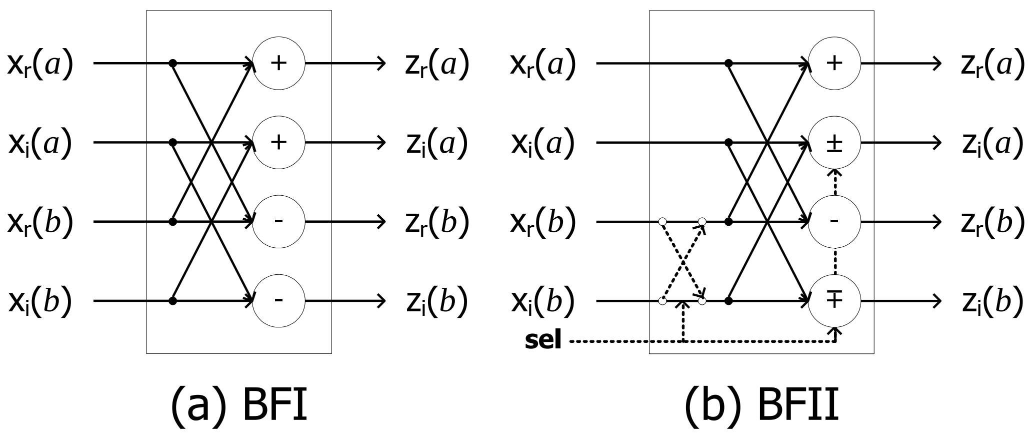


Figure 12 Two types of butterfly structures

two types of butterfly structures, BFI and BFII [41], are in use, as shown in Figure 12. BFI is basically composed of two complex adders/substractors for two complex inputs, *a* and *b*, while BFII contains additional multiplexing logic to implement an optional multiplication of –*j*; that is, the trivial twiddle factor multiplication of –*j* can be accomplished by a simple real-imaginary swap plus well-multiplexed addition/subtraction computations instead of actually using a costly complex multiplier.

Meanwhile, the formal definition of IPM $\boldsymbol{I_n}$ is described in (1), where *p* and *q* indicates the input and output port position respectively; and Figure 13 gives the examples of $\boldsymbol{I_4}$ and $\boldsymbol{I_8}$.

$$I_n : \begin{cases} q = p + (p \bmod 2) \times (\frac{n}{2} - 1), & p < \frac{n}{2} \\ q = p + (p \bmod 2) \times (\frac{n}{2} - 1) - (\frac{n}{2} - 1), & p \geq \frac{n}{2} \end{cases} \tag{8}$$

As a result, an IPM is simply a signal wiring network and thus hardly consumes logic resources from a hardware implementation perspective.

3.2.2 Hardware cost and throughput evaluation

For the conventional R2^{2}MDC architecture, the number of complex multipliers and adders is $2\lceil \log_4 N \rceil - 2$ and $2\log_2 N$, respectively. Besides, two output values

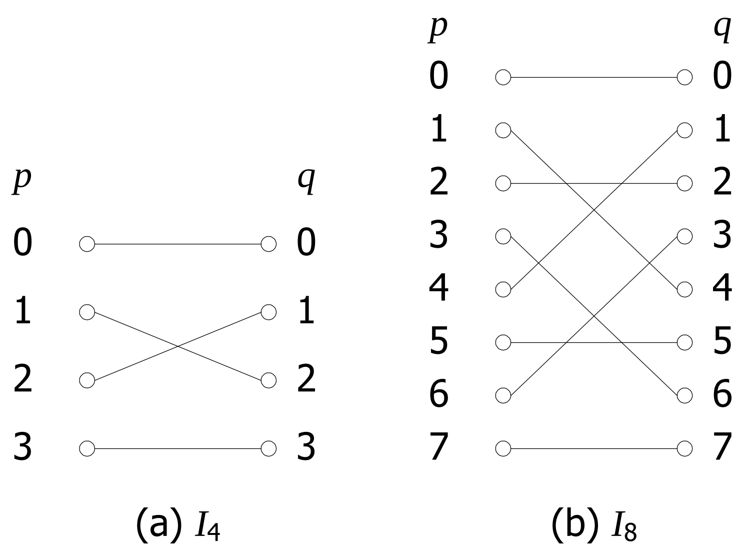


Figure 13 Interconnection configuration of I_4 and I_8

are produced at every cycle, so the throughput is 2/N. Furthermore, since R2^2EMDC allows expanding its datapath to trade for better performance, the number of multipliers, adders, and so as the overall throughput is all proportional to the degree of parallelism t. For example, Figure 14 illustrates four instances of the 16-point R2^2EMDC architecture with different t settings. Figure 14(a) shows the case with t = 1 (no expansion applied), which is identical to the original R2^2MDC architecture. Meanwhile, Figure 14(b)/(c) gives the case with t = 2/4, where the number of multipliers and adders is doubled/quadrupled, and so is the throughput. Notice that the number of FIFO entries actually decreases as t rises. Figure 14(d) depicts the case with t = 8, namely, the fully-expanded implementation that provides the maximum throughput of 16-point R2^2EMDC architecture, which also demands the largest hardware resources.

Similarly, for the foldable Pease architecture illustrated in Figure 10, it is observed that the number of multipliers and adders is both reduced to a half and so is the throughput as the datapath is vertically folded once. However, the number of FIFO entries remains unchanged regardless of the parallelism of datapath.

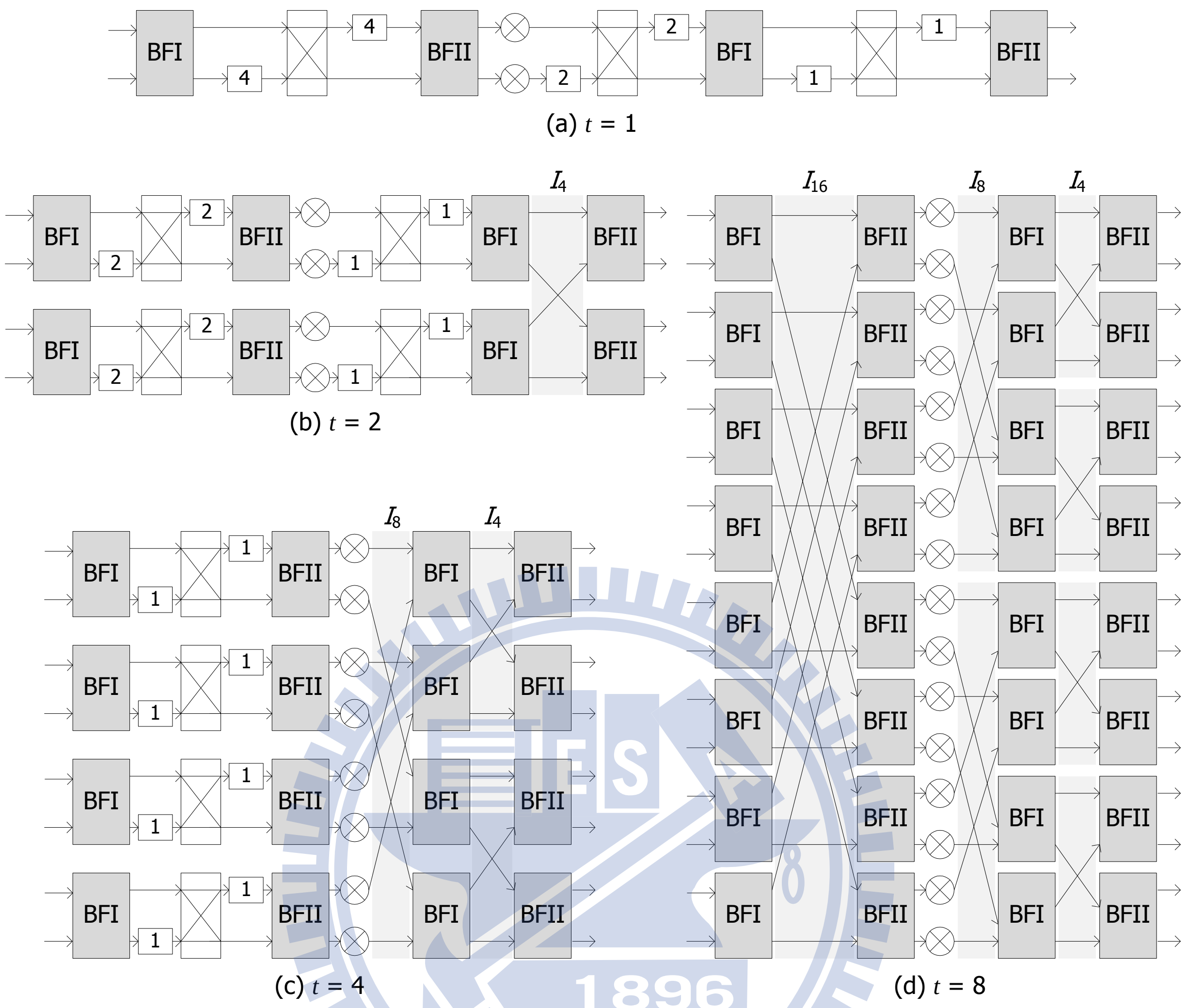


Figure 14 Different instances of 16-point R2^{2}EMDC architecture

Table 7 gives several theoretical comparisons between the existing foldable Pease architecture and the proposed R2^{2}EMDC one. Since multipliers take a major part of hardware cost in an FFT design, it is evident that our newly proposed one requests less hardware resources than the existing one under the same throughput constraint.

3.3 Experimental Results

Based on the proposed R2^{2}EMDC architecture, we have implemented a parameterizable FFT generator in Perl script. By indicating the size (N) of FFT core and the degree of datapath parallelism (t), our generator can output the specified hardware design in synthesizable Verilog HDL just in few seconds. Furthermore, we also use MATLAB in our testbench environment for extensive pattern generation/simulation and SQNR analysis (larger than 80db) to verify the correctness of the generator.

In additional to comparing the foldable Pease architecture and R2^{2}EMDC in an analytic way (Table 7), it is also interesting to compare the generated hardware cores in terms of logic-gate-count (NAND2-equivalent), throughput, and power consumption. As a result, we have also implemented the generator proposed in [43], which always completely folds the core in horizontal direction first and then v-folds the core in vertical direction, where v is parameterizable. The generated hardware cores are then synthesized under UMC 0.18um technology using Synopsys Design Compiler with 100MHz timing constraint.

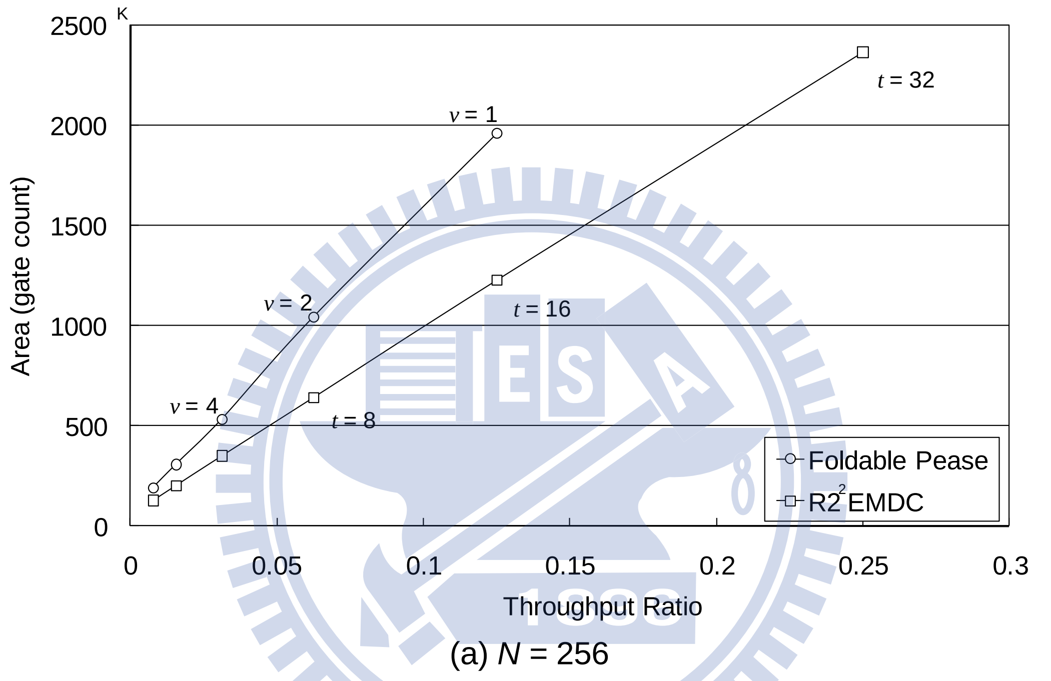
Figure 15(a) and (b) reports the area-throughput design tradeoff solutions offered by R2^{2}EMDC and the foldable Pease architecture [44] for 256-point and 1024-point FFT, respectively. The throughput ratio indicates the throughput of a given architecture to that of the fully parallel Pease architecture. As expected, the newly proposed one is always more area-efficient than the existing one when delivering the same throughput. To name an example, the area reduction is about 37% as N = 256 and t = 16. Moreover, under a fixed FFT size N, the area gap between two architectures is getting large as the target throughput increases. That is, according to

Table 7 Comparisons between foldable Pease and R2^{2}EMDC

Architecture	#multipliers	#adders	#FIFOs
Foldable Pease [54, 55]	$t\log_2 N$	$2t\log_2 N$	N
R2^{2}EMDC	$t(2\lceil\log_4 N\rceil - 2)$	$2t\log_2 N$	$N-2t$

the theoretical analyses and experimental results, it is conclusive that the proposed R2^{2}EMDC architecture is indeed capable of providing a smaller hardware implementation than the existing foldable Pease architecture under the same throughput constraint.

Table 8 compares the power consumption between four different instance pairs



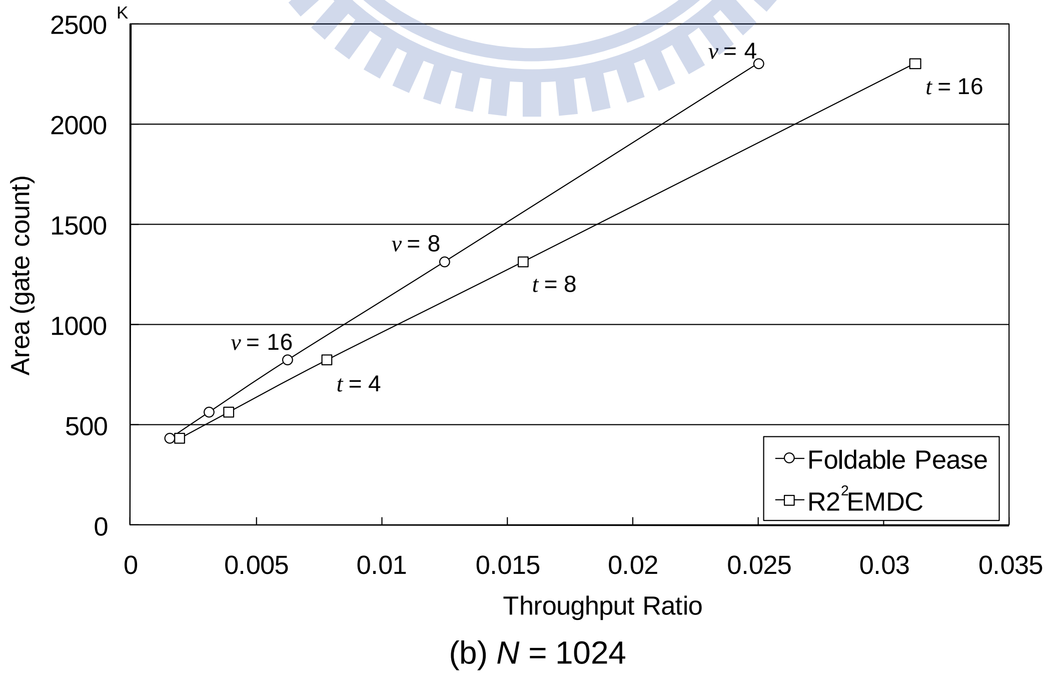


Figure 15 Area vs. throughput in 256/1024-point FFT

Table 8 Power consumption in 256-point FFT (mW)

Ratio	Foldable Pease	R2^{2}EMDC	Power Reduction (%)
0.0078	14.03	12.48	11.0
0.0156	27.59	22.09	20.0
0.0313	55.90	42.07	24.7
0.0625	97.67	79.41	18.7

derived from the two architectures with the same target throughput ratio (using UMC 0.18um process and measured by Synopsys PrimePower). R2^{2}EMDC achieves roughly 20% of power reduction as compared to the foldable Pease architecture. Here, the smaller hardware implementation should be the major reason for the results.

3.4 Summary

In this dissertation, we propose an expandable multi-path delay commutator (EMDC) based FFT architecture. We show that the proposed architecture can be easily and flexibly expanded to satisfy throughput-hungry applications. In addition, a parameterizable hardware generator is also developed to automatically produce the specified HDL code so that the design cost and time can be drastically minimized. Finally, the theoretical analyses and/or experimental results demonstrate that the proposed architecture does consume less area and power than the existing foldable Pease architecture under the same throughput constraint.

Chapter 4

Probability-Based Static Scaling Optimization for Fixed Wordlength FFT Processors

4.1 Introduction of Scaling Optimization

Recently, high-data-rate wireless communication systems have been becoming a primary focus of both research and development. For example, OFDM technology is one of the favorable choices for future broadband systems and is highly suitable for video transmission and mobile Internet applications. The FFT processor is one of the key components in OFDM-based wireless systems [4-6, 45]. Actually, not just in OFDM, the FFT processor also serves as an essential element in many other modern DSP systems. Consequently, improving FFT processor designs has become the focus of a large number of studies since the past decade.

The architectures of FFT processors can be roughly divided into two major categories: memory-based and pipeline-based. A memory-based architecture usually consists of only one butterfly unit. In general, it provides area-efficient solutions for low-throughput applications. Alternatively, a pipelined architecture consists of multiple concurrent processing butterfly units so that it can provide higher throughput at the cost of more hardware resources. In general, memory-based architectures are suitable for FFT processors where the hardware cost is an issue and the FFT size is not smaller than 512 [3]. Pipeline-based architectures are typically feasible for applications with smaller FFT sizes. In this dissertation, we focus on the scaling optimization on FFT designs with a **fixed wordlength** at each stage; that is, the output wordlength of every stage is the same as its input wordlength. Memory-based designs naturally fulfill this requirement, while a considerable part of pipeline-based designs also choose to meet the same requirement since the fixed wordlength is still preferred due to hardware cost and critical-path delay considerations [46].

While crafting a practical FFT hardware design, the output precision in terms of Signal-to-Quantization-Noise Ratio (SQNR) is regarded as a key design requirement. In practice, FFT algorithms are commonly implemented using fixed-point arithmetic instead of floating-point arithmetic for hardware cost reduction. That is, only a limited number of bits are available to represent a signal or coefficient value. As a result, rounding and truncation operations inevitably introduce noises, which are referred to as quantization noises. Besides, addition and subtraction operations may also cause overflow errors (noises) during computations. Although extending the wordlength can relieve the accuracy loss, the hardware cost and the critical-path delay are increased accordingly.

Therefore, several number scaling methods, either static or dynamic, have been proposed to improve the output SQNR [1, 45-69]. Oppenheim et al. [49] proposed a simple static scaling method that always increases the integer part by one bit for each radix-2 stage to prevent overflows. However, this method suffers significant quantization errors if the wordlength is fixed. In addition, methods based on dynamic scaling have also been proposed for the SQNR improvement. Block Floating Point (BFP)-based methods employ intermediate buffers to store and analyze a block of output values, and then dynamically determine an appropriate number format for that block to achieve a better SQNR [1, 45, 46, 56, 57]. However, all of these dynamic methods suffer a notable increase in area, power, and latency as well as need a more complicated control unit. Consequently, most FFT designers rely on static instead of dynamic scaling optimization techniques to determine a proper number format for each stage [50].

Previous static scaling techniques can be roughly classified into three major categories: simulation-based approaches [58, 59], analytical approaches [47-50, 60,-68], and a hybrid of previous two [48, 69, 70]. The simulation-based approaches try to find a good number format through lengthy iterations. In contrast, the analytical methods can determine a good number format very efficiently through a static numeric analysis without invoking time-consuming simulation. However, the analysis results are generally too pessimistic and lead to a larger wordlength than required. Therefore, the hybrid approaches are proposed to determine the number format and shorten the simulation time simultaneously. Meanwhile, the works mentioned above [47-50, 58-70] all assume they can arbitrarily determine the wordlength at each stage. However, in memory-based FFT designs, the wordlength (the width of the memory block) is always fixed. To the best of our knowledge, the problem of static scaling under a fixed wordlength constraint has not been well addressed yet.

In this dissertation, we propose a scaling optimization technique based on the static probability analysis that can rapidly determine the best number format at each butterfly stage under a fixed wordlength constraint. Given a probability distribution of input signals, a selected FFT algorithm, and a wordlength constraint, the proposed technique can maximize the overall output SQNR through the static number format analysis and optimization stage by stage. Compared to previous works, our method offers the following three contributions: 1) providing a probability model that can abstract the behavior of fixed-point arithmetic logic; 2) preventing the use of time-consuming and pattern-dependent simulation throughout the entire optimization process; 3) minimizing the required wordlength in a hardware implementation under a given SQNR target without demanding extra hardware components and complicating control logic compared to other existing static approaches [49, 50].

4.2 Number Scaling

4.2.1 Related works

It requires n+1 bits to accurately preserve a result of an n-bit fixed-point addition/subtraction operation. Hence, one solution for avoiding an overflow generated from a butterfly operation is to make the output wordlength one bit larger than the input one [48]. However, increasing the wordlength induces a number of drawbacks in FFT hardware implementation. First, a larger data storage unit (memory block or register file) is required, which increases both chip area and power consumption. Second, a longer wordlength results in a worse critical-path delay in arithmetic logic, which is not eligible for high-throughput FFT designs. Most of all, the wordlength is fixed in a memory-based FFT architecture, meaning that it is not possible to vary the wordlength from stage to stage. Consequently, many number scaling approaches have been proposed to prevent a wordlength increase at the cost of a minor accuracy loss, which can be roughly divided into two categories: the static scaling approaches and the dynamic ones.

Oppenheim et al. [49] proposed a static scaling procedure which is widely adopt in today’s FFT hardware implementation. Since the maximum magnitude of the result increases no more than a factor of 2 for a butterfly stage, incorporating an attenuation of 1/2 at both inputs (that is, increase the integer part by one bit and decrease the fractional part by one bit in a fixed-length word) to a radix-2 butterfly unit can completely eliminate output overflows. However, this approach degrades the output

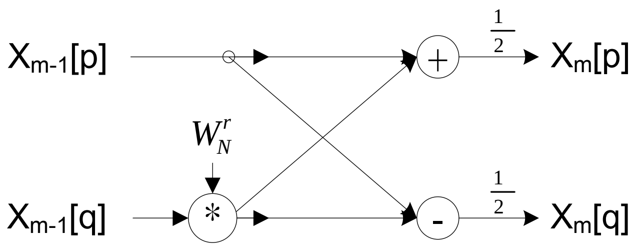


Figure 16 A radix-2 butterfly unit with scaling by 1/2 at the output.

SQNR due to larger truncation errors caused by the increasingly shorter fractional part stage by stage. Besides, the above scaling method can be further improved a bit with only a slight modification. Instead of performing number scaling at the input, incorporating an attenuation of 1/2 at the output of each stage, as shown in Figure 16, can achieve a better overall SQNR.

In [50], Ramakrishnan et al. concentrated on FFT designs for OFDM receivers. The authors exploit the fact that input samples of OFDM follow a normal distribution to predict the possible output value range at each stage and then determine the scaling strategy accordingly. They suggest increasing the integer part by one bit for every two stages instead of every stage for FFT designs used in OFDM. However, the input can vary from application to application, and is mostly assumed uniformly distributed in a typical FFT analysis [48]. Furthermore, our experimental results show that the approach presented in [50] works well only if the standard deviation of normal distribution is within a specific range.

Therefore, instead of adopting the methods proposed in [49] and [50] directly, most designers try to find the optimized number format of output for each stage through simulation if a better SQNR is expected. Typically, there are two options for determining the number format of a radix-2 butterfly stage: keeping it unchanged as at the previous stage, or moving one bit from the fractional part to the integer part. However, when the number of stages (k) is big due to a large FFT size, it is virtually impossible to evaluate all feasible configurations (2^k) and then pick the best one through simulation. Consequently, designers usually empirically select a limited set of "better" candidate configurations, and choose the best one among them still through extensive time-consuming simulation.

On the other hand, a dynamic scaling approach improves the output SQNR by means of the notion of shared-exponent. The BFP algorithm [1], which is one of dynamic scaling methods, employs an intermediate buffer to store a block of output data, detects the maximum value, and then determines the exponent for that block of data. Though this method does achieve a better result than common static scaling approaches, the extra data buffer implies a notable increase in area. As well, buffer access and exponent detection operations require longer processing latency and consume more power. Therefore, static scaling approaches are still much more commonly preferred for typical FFT hardware implementations.

In this dissertation, we propose a fast probability-based static scaling optimization technique that is capable of providing a better output SQNR than existing static ones as well as needs no simulation at all. It is also as area-efficient as other static methods since all of them do not require a dynamic scaling unit; however, our technique can still roughly achieve the same level of output quality when compared with dynamic scaling approaches. For every butterfly stage, the proposed method can precisely estimate the accuracy loss of each candidate number format due to possible saturation and truncation errors via the static probability-based analysis and then picks the best one of them. Furthermore, our method can work with various FFT sizes, FFT algorithms, wordlengths, and input signal distributions.

4.2.2 Motivation and problem definition

As mentioned, the approach proposed in [49] suggests increasing the bitwidth of the integer part by one at every radix-2 butterfly stage to avoid overflows. In this dissertation, the format for a fixed-point number is represented in the form of *m***b***n***f**, denoting the wordlength is *m*-bit, the integer part is *n*-bit and thus the fractional part takes the rest *m*-*n* bits. Note that an *m*-bit number can only represent 2^m different values no matter what the value of *n* is. Though the maximum magnitude of representable values can be doubled as increasing *n* by one, the bitwidth of the fractional part must be decreased by one at the same time since the wordlength *m* is fixed, which inevitably results in a precision loss. Take the radix-2 64-point FFT, which has $\log_2 64$ (i.e., 6) butterfly stages, as an example, if the input data is in 12**b**1**f** format, the final output format becomes 12**b**7**f**, in which only 5 bits are available for the fractional part.

However, after a 12-hour simulation with about 20 million random sets of input data, the probability for an output value that actually needs the seventh bit of the integer part is almost zero (i.e., the increasing one bit in the integer part is almost unnecessary). The fact implies that it may not be a wise method to always move a bit from the fractional part to the integer part at every butterfly stage since keeping more bits for fractional part can help reduce the truncation errors and thus improve the final output SQNR.

Nevertheless, if a stage keeps its output number format the same as its previous stage, then overflows might occur. In such cases, saturation logic is typically employed for overflow error reduction. A saturation operation is to clamp an overflowed positive/negative value to the maximum/minimum value a number format can hold. For example, if the number format in use is 4**b**4**f**, then 0100 (4) + 0101 (5) = 1001 (-7), which is an overflow with an error of 9 – (-7) = 16. However, if saturation is applied, the result becomes 0111 (7) and the error can be reduced to 9 – 7 = 2, which is much smaller than 16.

Let's examine a few configurations of the output number format for the 256-point FFT design with the input number format of 12**b**1**f**. If the configuration suggested in [49] is used, the resultant SQNR is 35.39 dB by simulation. If the integer part is not increased at the output of the 8th stage and saturation is performed**,** the SQNR would climb to 37.03 dB. If applying it again to the 7th stage, the SQNR would rise to 38.47dB. However, if further applying it to the 2nd stage, the SQNR would dramatically drop to 17.82 dB. The above indicates that the output number format of each stage must be determined carefully for achieving an even better SQNR.

Conventionally, static scaling optimization methods usually rely on simulation to evaluate the performance of a configuration, like [50]. Nevertheless, it takes hours for simulation with only ten thousand sets of inputs just to evaluate one single configuration of the 8192-point FFT. Meanwhile, for the radix-2 N-point FFT, there are $\log_2 N$ stages and the integer part can be increased by one bit or not at each stage, which makes the total number of possible configurations equal to N. That is, it takes years if one attempts to evaluate all configurations of the 8192-point FFT. This is the prime reason that motivates us to develop a revolutionary simulation-free scaling optimization technique, which turns out to be able to discover a near-optimal solution within only few minutes.

At the end of this section, the problem of static scaling optimization on fixed-point FFT addressing in this dissertation is described as follows – given FFT size, radix r (where r is a power of 2), fixed wordlength, and input probability distribution, determine the number format for the output of every stage statically (i.e., without use of simulation) such that the overall SQNR is maximized.

4.3 Probability Model

4.3.1 Probability-based SQNR analysis

The input to an FFT is modeled as a *discrete random variable*, which can take only a countable number of distinct values. A discrete random variable X has an associated *probability mass function* (PMF) [76, 77], denoted as p_X, which gives the occurrence probability of each possible value that X can take. In particular, if x is a possible value of X, the probability mass of x, denoted as $p_X(x)$, gives the probability that X is exactly equal to x, or

$$p_X(x) = P(\{X = x\}) \tag{9}$$

Also note that

$$\sum_x p_X(x) = 1 \tag{10}$$

where in the summation, x ranges over all the possible values of X.

Derived distribution is the PMF of a function of random variables with known distribution [63, 64]. The notion of derived distribution helps establish the following properties.

***Property* 1**: Suppose A and B are two independent discrete random variables; for each of them, the PMFs of its real and imaginary part are reflection-symmetric (w.r.t. 0) and identical (i.e., $p_{R(A)} = p_{I(A)}$ and $p_{R(B)} = p_{I(B)}$). Then, the PMF of $Z = A \; op \; B$ (*op* can be either addition or subtraction) can be derived by the convolution of the PMFs of A and B. As well, the PMFs of Z's real and imaginary part are also identical (i.e., $p_{R(Z)} = p_{I(Z)}$) and reflection-symmetric.

***Property* 2**: Assume A is a complex random variable and the PMFs of its real and imaginary part are identical and reflection-symmetric; w is a complex constant value. If $Z = A \cdot w$, then the PMFs of Z's real and imaginary part are also identical and reflection-symmetric.

Proof: Assume the PMF of A's real (imaginary) part is P and $w = a + bi$, then the PMFs of Z's real and imaginary part are $p_{R(Z)} = (aP - bP)$ and $p_{I(Z)} = (aP + bP)$, respectively. Since P is reflection-symmetric, $p_{R(Z)} = p_{I(Z)} = (a + b)P$. Hence, the PMFs of Z's real and imaginary part are identical and reflection-symmetric.

***Property* 3**: Given two complex numbers $X = R(X) + jI(X)$ and $Y = R(Y) + jI(Y)$, where $p_{R(X)} = p_{I(X)}$, $p_{R(Y)} = p_{I(Y)}$ and all four PMFs are reflection-symmetric. Then, for the radix-2 butterfly operation $Z = BF(X, Y)$, the PMFs of Z's real and imaginary part are also identical and reflection-symmetric according to *Property* 1 and *Property* 2. That is

$$p_{R(Z)} = p_{I(Z)} \tag{11}$$

***Property* 4**: Since $Y = FFT(X)$ is composed of a series of radix-2 butterfly operations, the PMFs of Y's real and imaginary part are also identical and reflection-symmetric according to *Property* 3, or

$$p_{R(Y)} = p_{I(Y)} \tag{12}$$

***Property* 5:** For a signal Y, if the PMFs of Y's real and imaginary part are identical, then the power of Y is two times larger than the power of its real (imaginary) part based on (12), or

$$
\begin{aligned}
Power_Y &= \sum\nolimits_y (y^2 \cdot p_Y(y)) \\
&= \sum\nolimits_y (R(y)^2 + I(y)^2) \cdot p_{R(Y)}(R(y)) \cdot p_{I(Y)}(I(y)) \\
&= \sum\nolimits_y R(y)^2 \cdot p_{R(Y)}(R(y)) \cdot p_{I(Y)}(I(y)) + \\
&\quad \sum\nolimits_y I(y)^2 \cdot p_{R(Y)}(R(y)) \cdot p_{I(Y)}(I(y)) \\
&= \sum\nolimits_{R(y)} R(y)^2 \cdot p_{R(Y)}(R(y)) \cdot \sum\nolimits_{I(y)} p_{I(y)}(I(y)) + \\
&\quad \sum\nolimits_{I(y)} I(y)^2 \cdot p_{I(Y)}(I(y)) \cdot \sum\nolimits_{R(y)} p_{R(y)}(R(y)) \\
&= \sum\nolimits_{R(y)} R(y)^2 \cdot p_{R(Y)}(R(y)) + \sum\nolimits_{I(y)} I(y)^2 \cdot p_{I(Y)}(I(y)) \\
&= 2\sum\nolimits_{R(y)} R(y)^2 \cdot p_{R(Y)}(R(y))
\end{aligned}
\tag{13}
$$

***Property* 6:** In a fixed-point FFT, because both the real and imaginary part use the same number format at every stage, the noises induced from saturation and quantization are also identical and symmetric in both parts, i.e., for the noise E, $p_{R(E)}$ and $p_{I(E)}$ are identical. Similarly, the power of E is two times larger than the power of its real (or imaginary) part, or

$$Power_E = 2\sum\nolimits_{R(e)} R(e)^2 \cdot p_{R(Y)}(R(y)) \tag{14}$$

***Property* 7:** According to *Property* 5 and *Property* 6, the SQNR of $Y = FFT(X)$ is identical to that of its real (or imaginary) part if $p_{R(X)}$ and $p_{I(X)}$ are identical and reflection-symmetric, or

$$SQNR_Y = SQNR_{R(Y)} \tag{15}$$

In other words, *Property* 7 implies that we only have to consider the real (or imaginary) part while estimating the SQNR of an FFT implementation. Note that assuming $p_{R(X)}$ and $p_{I(X)}$ identical as well as reflection-symmetric for the input X of FFT is not uncommon; especially for today's digital communication systems (e.g., [78, 79]). The interleaved input signal of FFT can thus be considered as an identical and reflection-symmetric independent random variable. Therefore, the input signal to the

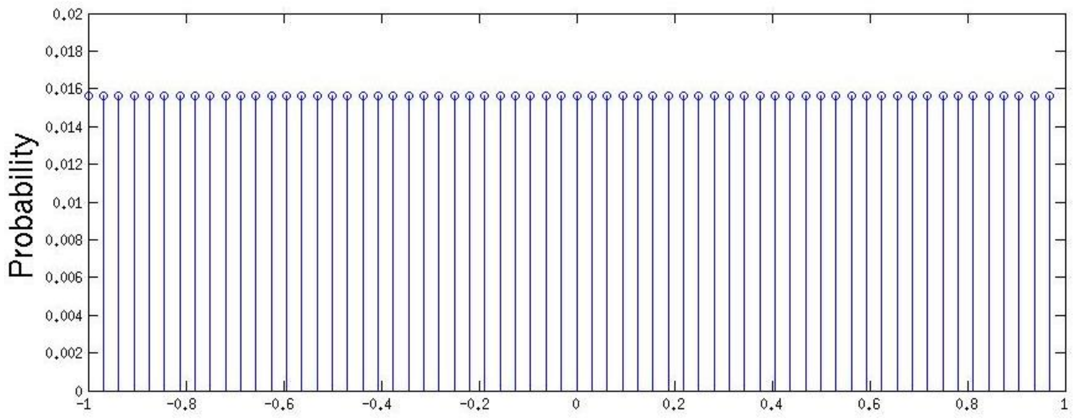


Figure 17 PMF of a 6-bit random variable with a uniform distribution.

FFT are commonly modeled with either the uniform distribution or the normal distribution (with μ=0) [48, 68].

Unless specified otherwise, either the real or imaginary part of the input signal is assumed a discrete random variable that is independent from each other and is uniformly distributed within [-1, 1) in the rest of this work. Figure 17 gives an example of the PMF for a 6-bit random variable with a uniform distribution.

4.3.2 Butterfly analysis

A radix-2 butterfly consists of two arithmetic operations: an addition (or subtraction) and a twiddle factor multiplication. For the addition operation $Z = A + B$, where A and B are two independent random variables with the PMFs p_A and p_B respectively, the PMF of Z can be given as

$$\begin{aligned} p_Z(z) &= P(a+b=z) \\ &= \sum_{\{(a,b)|a+b=z\}} P(A=a, B=b) \\ &= \sum_a P(A=a, B=z-a) \\ &= \sum_a p_A(a)\cdot p_B(z-a) \end{aligned} \tag{16}$$

The resulting PMF p_Z is called the convolution of the PMFs of A and B. Similarly,

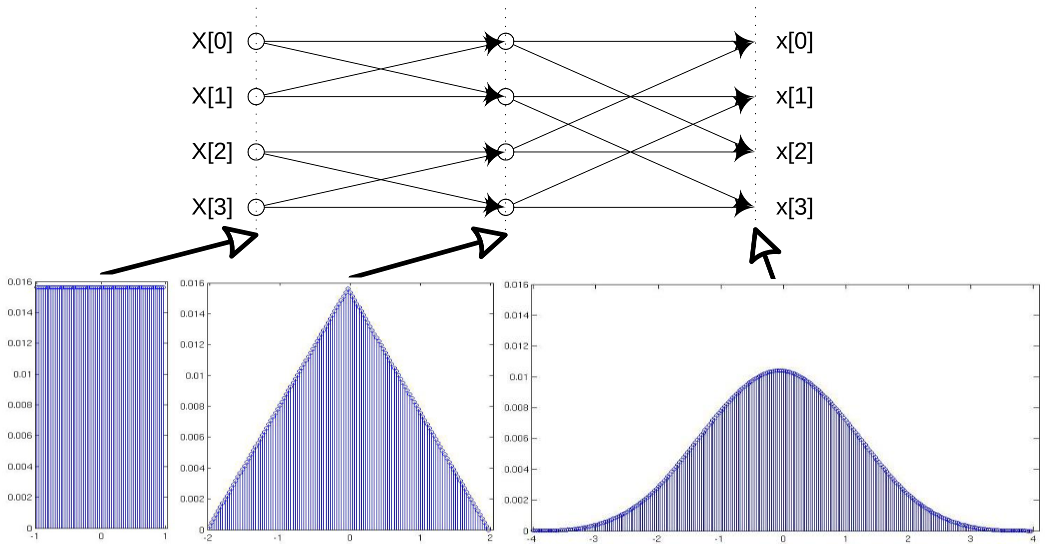


Figure 18 The derived distributions for the 4-point FFT.

the PMF of the subtraction operation can be derived in the same way.

Based on *Property* 3 and *Property* 4, the PMF of the FFT output can be obtained by iteratively applying (16) for a proper number of times. For instance, Figure 18 shows the PMF of the output at each stage for the ideal 4-point FFT where the x-axis shows the data value and the y-axis indicates the occurrence probability.

4.3.3 Saturation analysis

A saturation operation clamps the output between a maximum (*max*) and a minimum (*min*) representable value, which are determined by the given number format. For example, if the number format is 6**b**4**f**, (*max*, *min*) equals to (7.75, -8). For the saturation operation $Y = S(X)$, the PMF of Y can be calculated as

$$P_Y(y) = \begin{cases} \sum_{x \le min} P_X(x) & y = min \\ P_X(x) & min < y < max \\ \sum_{x \ge max} P_X(x) & y = max \end{cases} \qquad (17)$$

As previously mentioned, overflows may occur if the bitwidth of the integer part is not increased at a butterfly stage. The saturation logic eliminates those overflows and thus reduces computation errors (noises). When saturation is in use, (17) can help accurately model the probability distribution of the output.

4.3.4 Truncation analysis

A truncation operation approximates a fixed-point input value x with the output value trunc(x) by discarding a specific number of x's LSBs. For example, given a value of 011101 in 6b3f format (3.625), the truncated value in 4b3f format is 0111 (3.5) by discarding the trailing two bits. The truncation operation Y = T(X) generates the output value in *m*b*n*f number format with a *minimum scale* Δ of $1/2^{m-n}$, and thus the PMF of Y can be expressed as

$$P_Y(y) = \sum_{y \le x < y+\Delta} P_X(x) \tag{18}$$

As previously mentioned, truncations are mandatory if the bitwidth of the fractional part is decreased at a butterfly stage due to the fixed wordlength constraint. Then (18) is used to precisely model the probability distribution of the output after truncation.

4.3.5 SQNR estimation

To demonstrate the effectiveness of the proposed static probability-based analytical method, the output SQNR is estimated by both analysis and simulation. To estimate the output SQNR analytically, we need to know the PMFs of the ideal noise-free output X and the actual output $Y = g(X)$, where g can be truncation, saturation, or a combination of two. Assume the wordlength is unlimited so that neither overflow nor truncation may occur, the PMF of X (i.e., p_X) can then be obtained through derived distribution analysis. However, because the wordlength cannot be unlimited in a realistic implementation, the PMF of the actual output Y is very likely to be different from p_X due to saturation and truncation. Hence, the output SQNR can be estimated as

$$SQNR = 10 \cdot \log_{10}\left(\frac{Power_{signal}}{Power_{noise}}\right) \tag{19}$$

where the power of the ideal output signal X can be given as

$$Power_{signal} = \sum_x (x^2 \cdot p_X(x)) \tag{20}$$

and the power of the noise can be calculate as

$$Power_{noise} = \sum_x ((x - g(x))^2 \cdot p_X(x)) \tag{21}$$

Take the 64-point FFT with the uniformly-distributed input in 12**b**1**f** format as an example and assume the entire computation process is error-free except that the final output value is saturated and truncated to fit in a specific number format. Table 9 reports the SQNR and overflow probability for four different number formats. It is evident that the format 12b6f achieves the best SQNR among four candidates. The reason why 12**b**6**f** outperforms 12**b**5**f** and 12**b**4**f** is its extremely low overflow probability compared to the other two counterparts since overflows usually induce large errors even if saturation is employed. However, though there is no overflow in 12**b**7**f** at all, it still results in a lower SQNR than 12**b**6**f**. It is because the longer fractional part (i.e., smaller minimum scale) of 12**b**6**f** has a lower truncation error that can make up the minor accuracy loss due to saturation. Besides, the above analysis again suggests that increasing the integer part by one bit for every radix-2 butterfly stage is not always the best idea.

In addition to static analysis, simulation is also used to calculate the SQNR and overflow probability, and the comparisons are given in Table 10. For each output format, it takes about 12 hours for simulation with 21.6 million randomly generated input sets. Table 10 indicates that the difference in SQNR is very small (less than 0.15 dB) and the overflow probabilities estimated using the two methods are also extremely close. Therefore, it is clear that the probability-based analysis in minutes can achieve the same outcome against simulation in hours.

Table 9 Analysis for different output formats in 64-point FFT

Output format	Value range	Min. scale	SQNR (dB)	Overflow probability (%)
12**b**7**f**	[-64, 64)	0.0313	48.17	0
12**b**6**f**	[-32, 32)	0.0156	54.19	5.53E-11
12**b**5**f**	[-16, 16)	0.0078	42.49	4.86E-2
12**b**4**f**	[-8, 8)	0.0039	16.14	8.33

Table 10 Comparisons between analytical method and simulation method

Output format	Analytical method		Simulation method	
	SQNR (dB)	Overflow probability (%)	SQNR (dB)	Overflow probability (%)
12**b**7**f**	48.17	0	48.17	0
12**b**6**f**	54.19	5.53E-11	54.20	0
12**b**5**f**	42.49	4.86E-2	42.34	4.98E-2
12**b**4**f**	16.14	8.33	16.13	8.34

4.4 Scaling Optimization

In this section, we focus on how to make scaling decisions from stage to stage, and then propose our scaling optimization flow.

Since the wordlength is fixed, the number of representable values at each stage is always the same. Therefore, a scaling decision that determines the output number format has to be made at each stage. Based on the probability model and the derived distributions of butterfly, saturation, and truncation operations presented in Section 4.3, we further propose an algorithm that can efficiently evaluate different scaling options (i.e., number formats) at a stage and then determine the one with the highest output SQNR.

4.4.1 Scaling decision

Again, we first take the radix-2 FFT as an example. If the input format is *m***b***n***f** for some stage, it would be better to set the output format of that stage as (*m*+1)**b(***n*+1)**f** to minimize errors. However, in a fixed-wordlength butterfly implementation, the output can merely be represented using *m* bits, which leaves only two options: 1) moving one bit from the fractional part to the integer part (i.e., *m***b(***n*+1)**f**); or 2) keeping the output format the same as the input one (i.e., *m***b***n***f**). Apparently, the former results in no overflow but suffers more serious truncation errors, whereas the latter induces relatively smaller truncation errors but may experience saturation errors due to overflow. It is really hard to tell which one is better unless their corresponding SQNR values are known.

Instead of applying time-consuming simulation, the efficient probability-based techniques presented in Section 4.3 are used for SQNR estimation stage by stage. Assume the input format is *m***b***n***f** in the current stage and the PMF of the ideal noise-free output in (*m*+1)**b**(*n*+1)**f** format is p_I. Next, if *m***b**(*n*+1)**f** format is selected, the PMF of the truncated output p_T can be obtained through (18) and p_I. Similarly, if *m***b***n***f** format is selected, the PMF of the saturated output p_S can be obtained through (17) and p_I. Besides, the SQNRs for both options can also be calculated through (19)~(20). At last, the better number format can then be determined for the output of

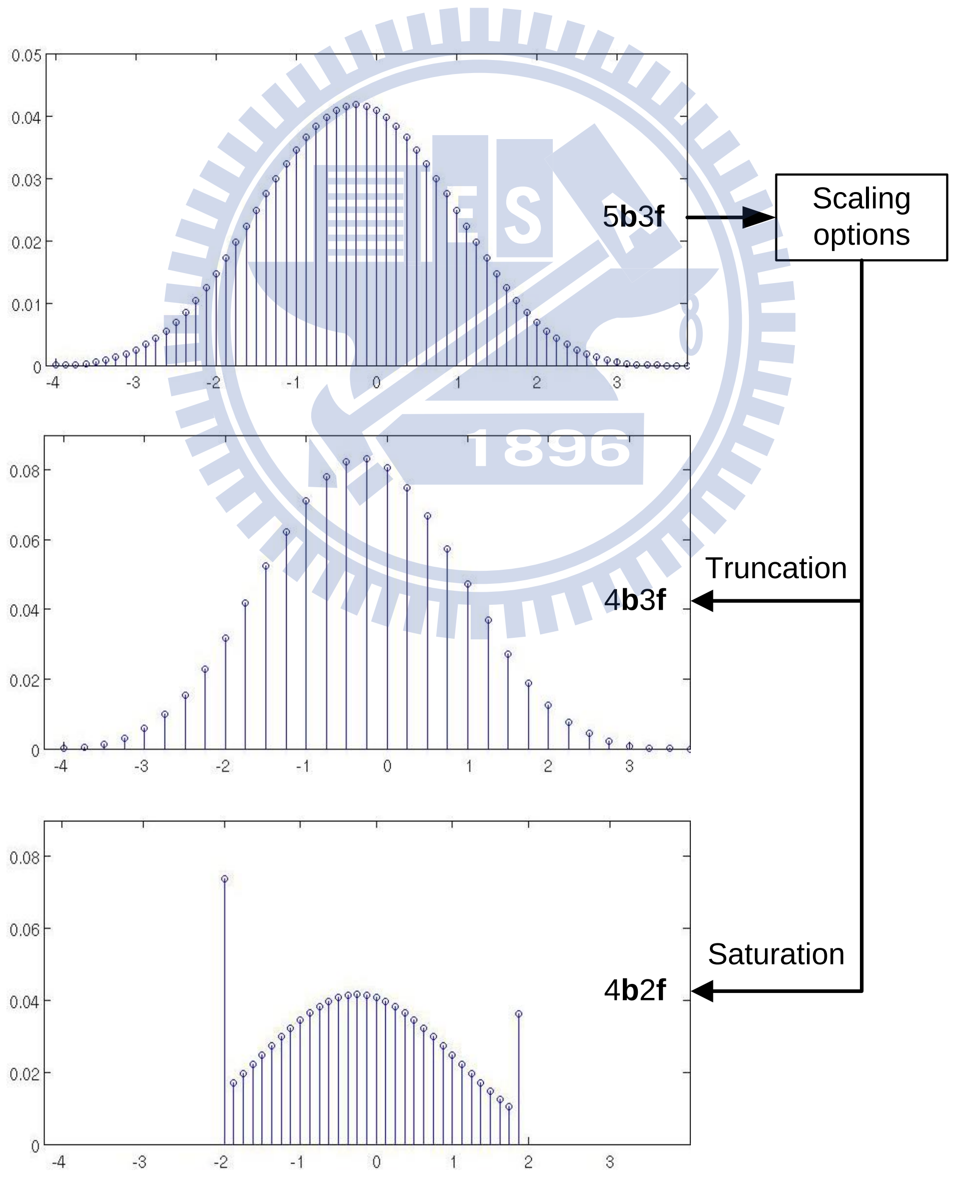


Figure 19 Differences in PMFs for two scaling options.

this stage and the corresponding PMF is used as the PMF of the input for the next stage. Figure 19 presents an example case illustrating the differences between PMFs of the two scaling options, in which the input is in 5**b**3**f** format.

The above procedure can certainly handle a butterfly with higher radix. For example, for a radix-4 butterfly stage, the output needs two more bits in the integer part to avoid overflows at all. Hence, there are three scaling options available for a radix-4 stage, i.e., moving 0~2 bits from the fractional part to the integer part. In fact, there are $(\log_2 r)+1$ scaling options for a radix-*r* stage. Nevertheless, no matter what the value of r is, our probability-based approach can always find the best choice (with the highest SQNR) among various candidates. Figure 20 gives the flow of the proposed scaling decision method for a radix-*r* stage.

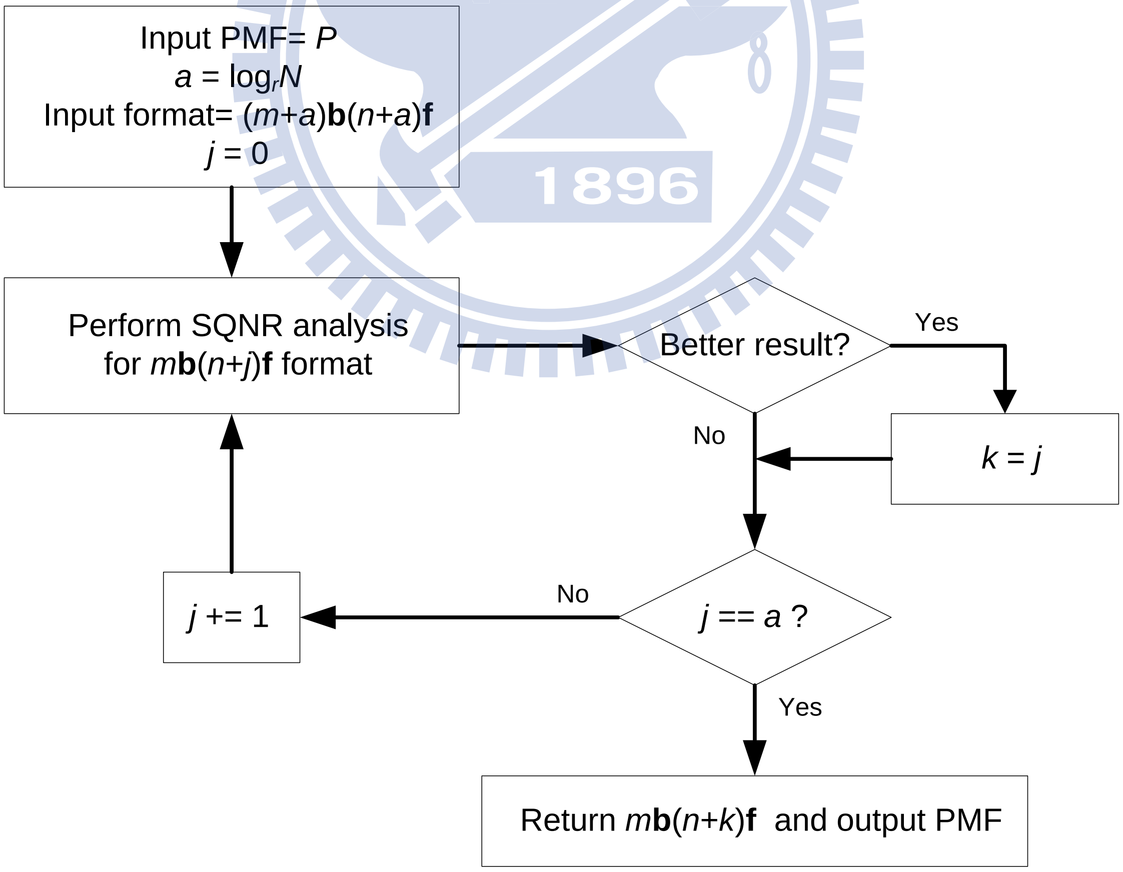

Input PMF= *P*
a = log*r**N*
Input format= (*m*+*a*)**b**(*n*+*a*)**f**
j = 0
Perform SQNR analysis for *m***b**(*n*+*j*)**f** format
Better result?
Yes
No
k = *j*
j == *a* ?
No
j += 1
Yes
Return *m***b**(*n*+*k*)**f** and output PMF

Figure 20 The proposed scaling decision flow for radix-*r* butterfly.

4.4.2 Scaling optimization flow

Figure 21 illustrates the overall flow of our scaling optimization algorithm. Given FFT size (N) and radix (r) of the FFT as well as number format (Q) and PMF (P) of the primary input, the proposed algorithm can find the best output number format for one stage at a time. At Stage s, the noise-free output PMF P' after a radix-r butterfly operation with the input PMF P can be derived using the methods presented in Section 4.3.2. Next, given P' and the fixed wordlength constraint, the techniques for scaling decision described in Section 4.4.1 can thus find out the best number format Q_s for Stage s as well as the corresponding output PMF P_s. If Stage s is not the last stage, then P_s and Q_s become the PMF and the number format of the input to the next stage. This process is not terminated until the output number format of every stage is determined.

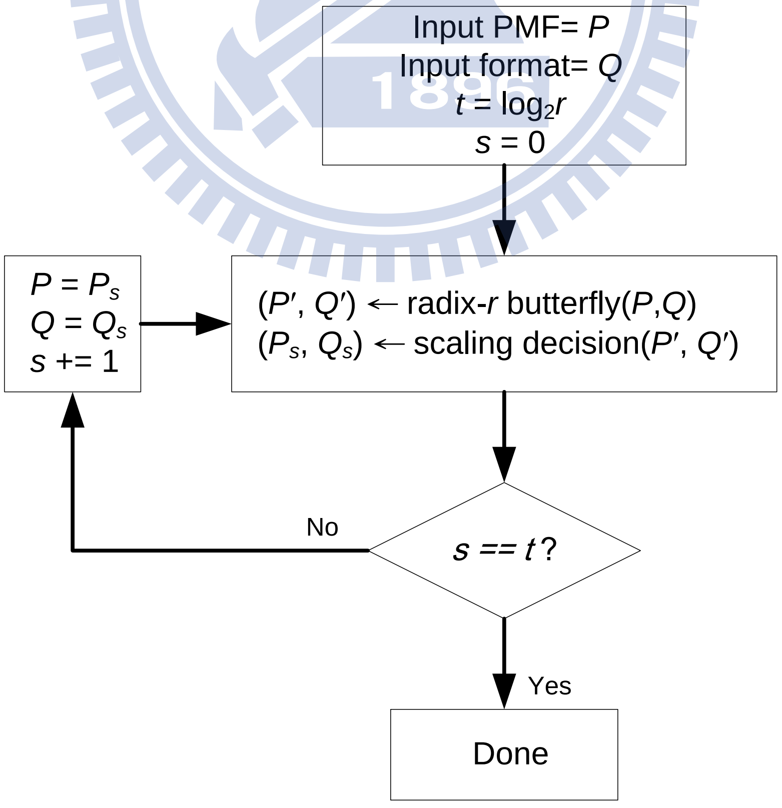

Input PMF= P
Input format= Q
$t = \log_2 r$
$s = 0$
$P = P_s$
$Q = Q_s$
s += 1
$(P', Q') \leftarrow$ radix-r butterfly(P,Q)
$(P_s, Q_s) \leftarrow$ scaling decision(P', Q')
No
$s == t$?
Yes
Done

Figure 21 The proposed scaling optimization flow.

The time complexity of the proposed algorithm is O($N_f \times N_s \times 2^w$) for the N-point radix-r FFT with a fixed wordlength w, where the number of candidate formats N_f equals to ($\log_2 r$)+1 and the number of butterfly stages N_s equals to $\log_r N$.

4.5 Experimental Results

The proposed scaling optimization algorithm has been implemented using MATLAB. Given FFT size, radix, fixed wordlength, number format of the input, PMF of the input, our tool can suggest the number format of the output for every butterfly stage in just a few minutes (even seconds).

To evaluate the effectiveness and efficiency of our scaling optimization algorithm, we have conducted a series of experiments and compare our results with those of other existing techniques. All experiments were conducted on a Linux workstation with an Intel dual Pentium Xeon 2.5GHz CPU and 32GB RAM

4.5.1 SQNR for varied configurations and sizes

Figure 22 shows the SQNR results for all 256 possible scaling configurations of the 256-point radix-2 FFT with a 12-bit wordlength, and the input is uniformly distributed within [-1, 1) in 12**b**1**f** format. It totally takes about 68 hours to complete the whole MATLAB simulation, in which the SQNR of each configuration is estimated by one thousand sets of randomly generated input data.

In Figure 22, a scaling configuration is encoded with an 8-bit ID indicating the corresponding scaling decisions at 8 consecutive butterfly stages. A '1' at the k^{th} most significant bit of an ID indicates the integer part of the output is increased by one bit at the k^{th} stage, while a '0' implies no format change. For example, in the configuration with ID = 245 (11110101), the bit counts of the integer part are 2, 3, 4, 5, 5, 6, 6, and 7 respectively from the first stage to the last. Figure 22 also reports that the

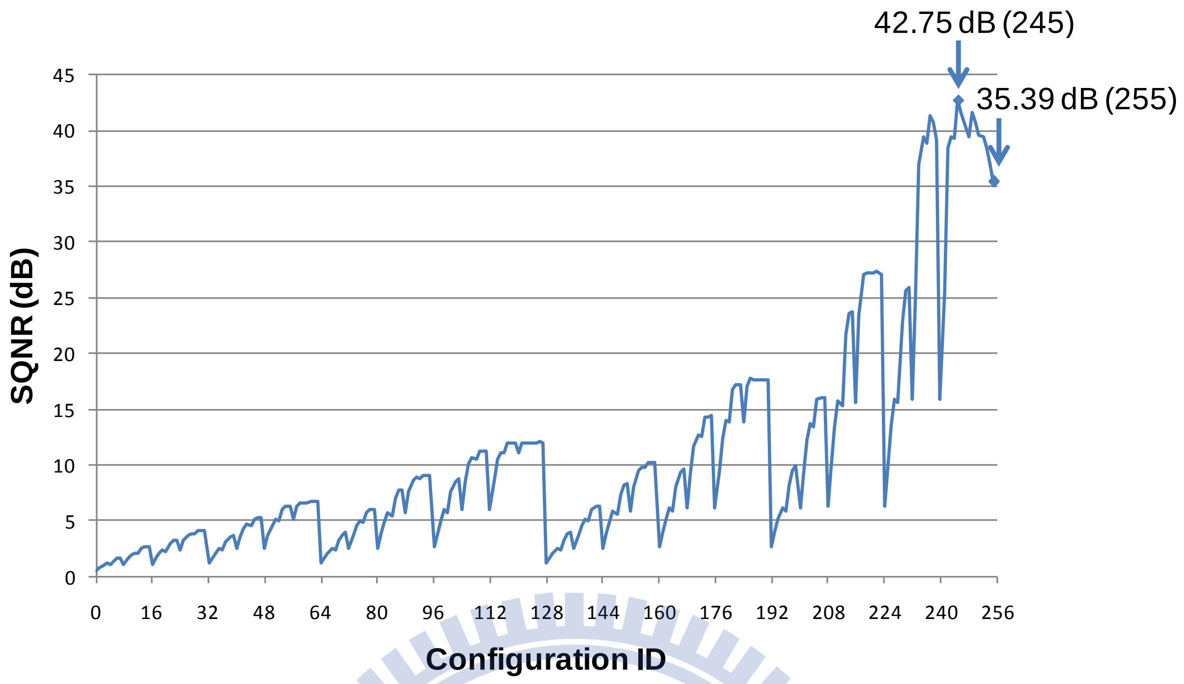
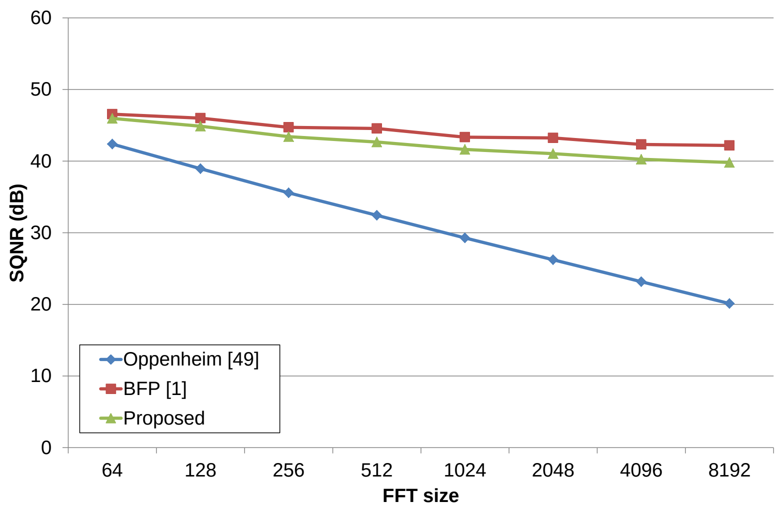


Figure 22 SQNRs of all scaling configurations for 256-point radix-2 FFT.

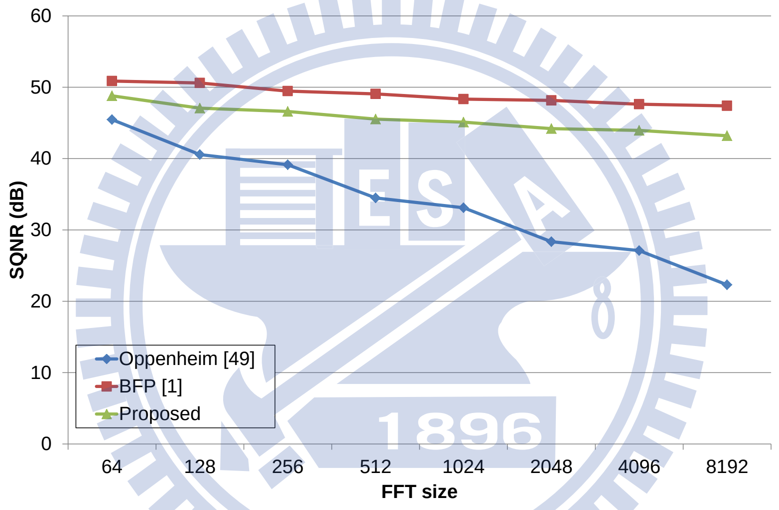
configuration with ID = 245, which is exactly the one selected by our algorithm, achieves the best SQNR of 42.75dB. That is, for the 256-point radix-2 FFT, our method merely needs seconds to identify the best solution while simulation requires days instead. This performance gap is getting bigger as the FFT size increases.

Furthermore, the configuration with ID = 255, which increases the integer part by one bit at every stage (i.e., the scheme suggested by Oppenheim et al. [49]) only achieves an SQNR of 35.39dB. It is also evident that a configuration with a larger ID tends to get a higher SQNR. The primary reason is that increasing the integer part at earlier stages effectively suppresses large errors induced from saturation. Nevertheless, though the Oppenheim's strategy generally results in a fairly good solution, it still leaves a big room for further improvement as Figure 22 suggests.

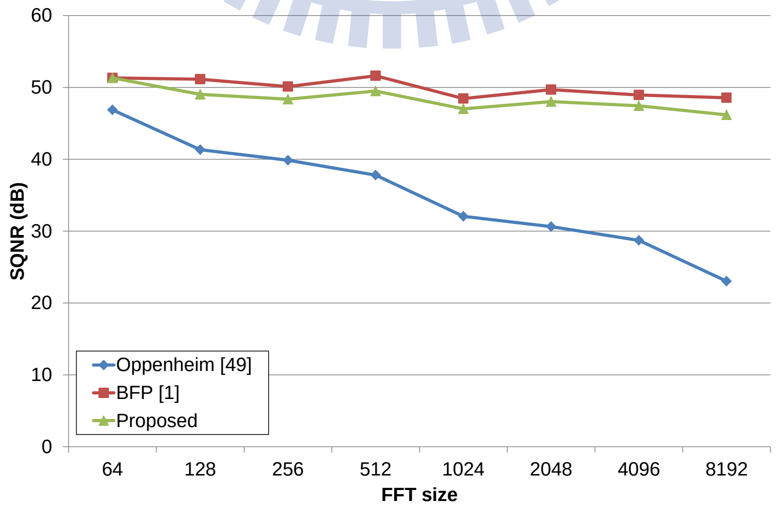
Figure 23 illustrates the SQNR comparisons among three different scaling approaches in terms of various FFT sizes and radices. Again, the input is uniformly distributed within [-1, 1) in 12**b**1**f** format. Figure 23 indicates that the output SQNR is always dropped as the FFT size increases due to the fixed wordlength constraint for all three methods. The Oppenheim’s method and ours are both static. Figure 23 shows that our approach always outperforms Oppenheim’s no matter what the radix is, and the advantage is getting bigger as the FFT size grows. The reason is that Oppenheim’s decreases the fractional part by one bit at each stage and thus suffers larger quantization errors than ours. Meanwhile, the BFP algorithm [1], a dynamic scaling technique, achieves slightly better results than ours. However, as mentioned before, a BFP-based implementation demands an extra hardware unit, which definitely makes a negative impact on area, latency and power consumption.



(a) Radix-2 FFT



(b) Radix-4 FFT



(c) Radix-8 FFT

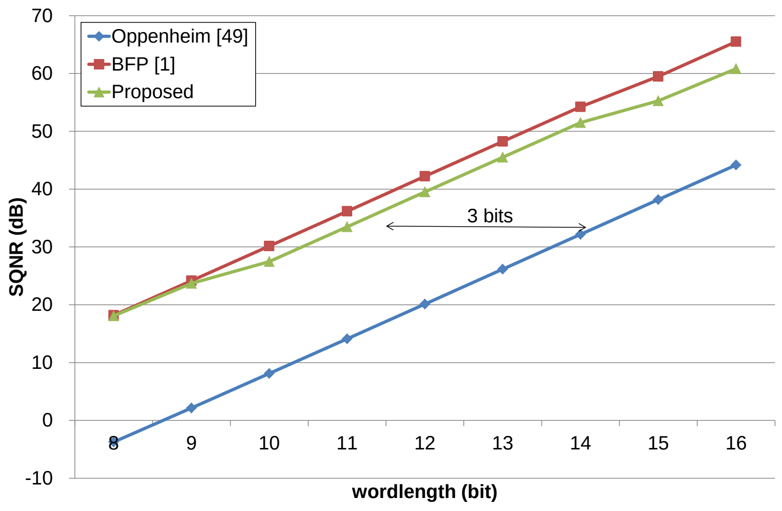
Figure 23 SQNR vs. FFT size (12 bits).

4.5.2 SQNR comparisons for varied wordlengths and distributions

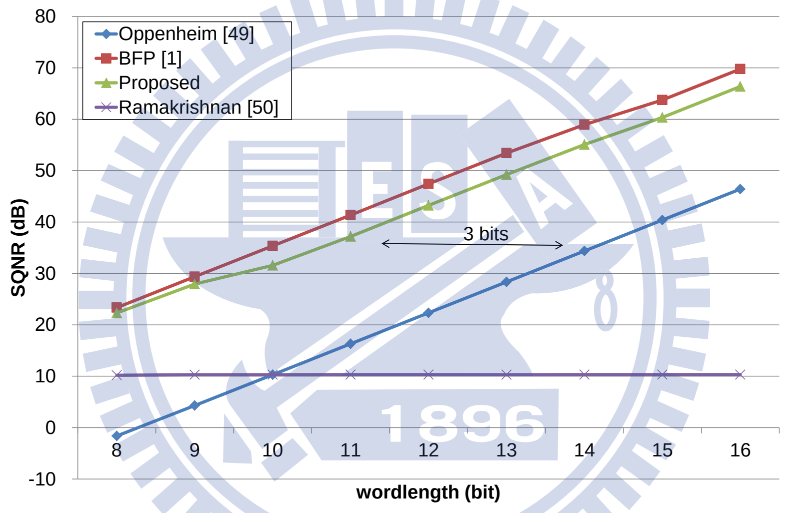
Table 11 presents the scaling optimization outcomes of our algorithm for the 8192-point radix-2 FFT with a varied wordlength *w*, and the input is still uniformly distributed in [-1, 1) in *w***b**1**f** format. Our algorithm merely requires 0.04 to 127.2 seconds to complete scaling optimization for the wordlength varying from 8 to 16 bits. Table 11 clearly shows that our algorithm significantly outperforms Oppenheim's [49] for all wordlength settings.

Table 11 Scaling optimization outcomes for 8192-point radix-2 FFT

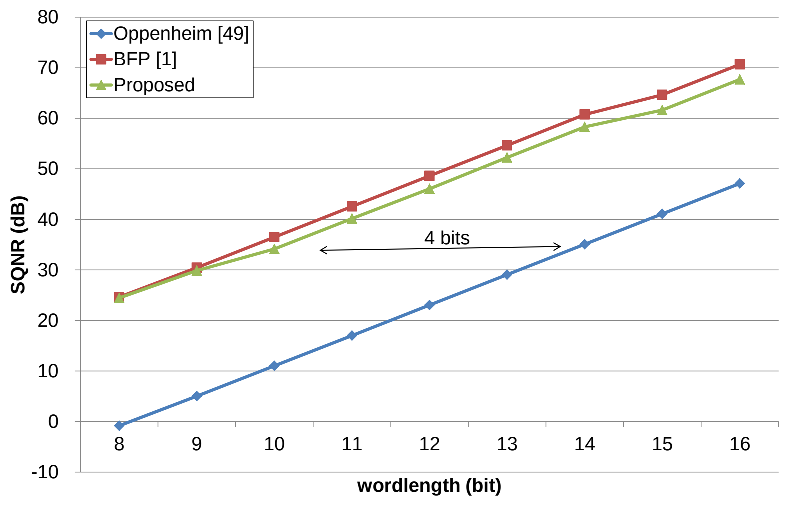
Wordlength (bit)	Integer bits of each stage													Runtime (second)	SQNR (proposed)	SQNR [49]
	1	2	3	4	5	6	7	8	9	10	11	12	13			
8	2	3	4	4	5	5	6	6	7	7	8	8	9	0.04	18.08 dB	-3.75 dB
9	2	3	4	4	5	5	6	6	7	7	8	8	9	0.06	23.70 dB	2.14 dB
10	2	3	4	5	5	6	6	7	7	8	8	9	9	0.07	27.47 dB	8.11 dB
11	2	3	4	5	5	6	6	7	7	8	8	9	9	0.11	33.47 dB	14.10 dB
12	2	3	4	5	5	6	6	7	7	8	8	9	9	1.60	39.50 dB	20.13 dB
13	2	3	4	5	5	6	6	7	7	8	8	9	9	3.52	45.51 dB	26.15 dB
14	2	3	4	5	5	6	6	7	7	8	8	9	9	18.36	51.50 dB	32.16 dB
15	2	3	4	5	5	6	7	7	8	8	9	9	10	44.44	55.28 dB	38.18 dB
16	2	3	4	5	6	6	7	7	8	8	9	9	10	127.20	60.83 dB	44.18 dB



(a) Radix-2 FFT



(b) Radix-4 FFT



(c) Radix-8 FFT

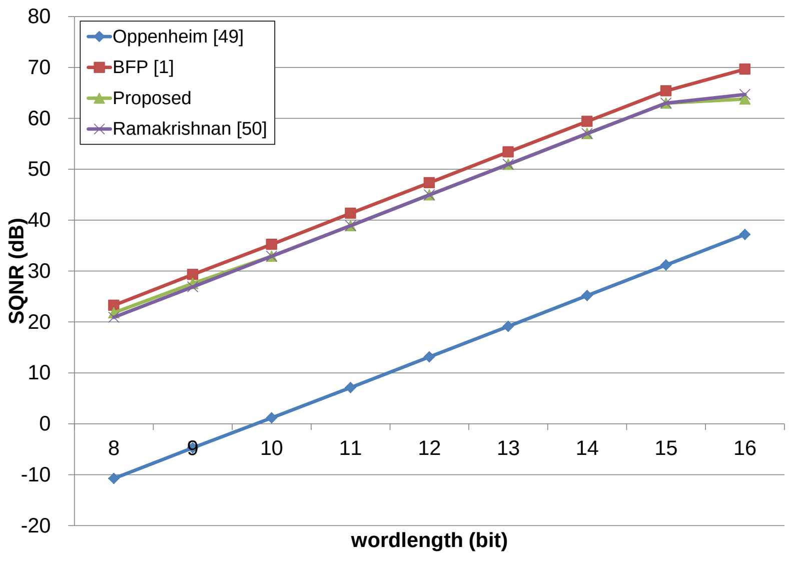
Figure 24 SQNR vs. wordlength (8192-point).

Figure 24 illustrates the SQNR comparisons among different scaling approaches in terms of wordlengths and radices. From Figure 24, it is observed that our approach can save about 3~4 bits of wordlength but still achieves a same SQNR target as compared with the Oppenheim's. To demonstrate what difference a wordlength of 3 bits can make, we implemented both the 11-bit and 14-bit 100MHz 8192-point radix-2 FFT hardware modules using TSMC 180nm technology and Synopsys DesignWare. Table 12 gives the comparisons between them as well as shows that the improvements are very significant in terms of area, power, and memory bits. In addition, Figure 24 indicates that our technique achieves a similar quality of result (QoR) for all combinations of various wordlengths and radices as compared to the BFP-based dynamic scaling approach [1].

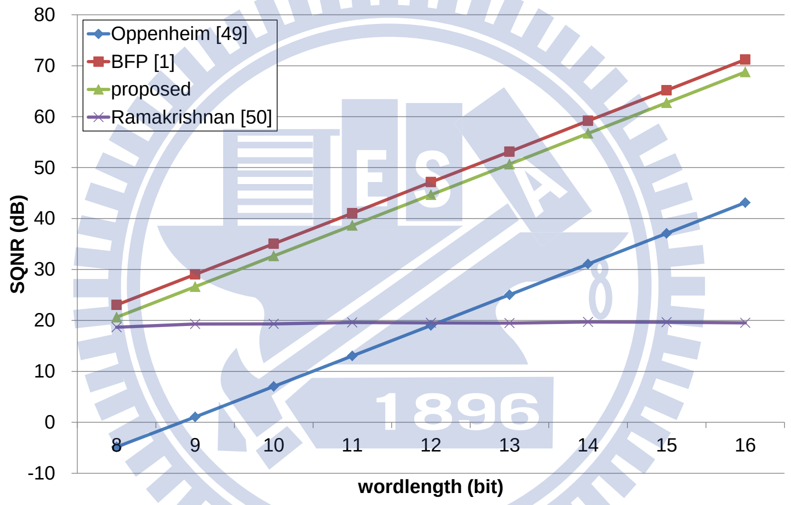
Figure 25(b) also shows that Ramakrishnan's approach [50] seriously underperforms than the others. It is because the Ramakrishnan's method expects a normally distributed input. Hence, we conducted another set of experiments on the 8192-point radix-4 FFT with the normally distributed input within [-1, 1). As shown in Figure 25, the SQNR results presents the standard deviation (σ) is set to 0.2 and 0.4. The Ramakrishnan's method [50] performs nearly as well as ours if $\sigma = 0.2$, whereas it again seriously underperforms if $\sigma = 0.4$.

Table 12 Hardware comparisons under the same SQNR constraint

8192-point radix-2 FFT (100MHz)	Proposed (11-bit)	Oppenheim [49] (14-bit)	Reduction
Area (μm^2)	85,771.2	129,852.7	33.65 %
Power (mW)	1.9229	2.6302	26.89 %
Storage (bit)	180k	229k	21.41 %



(a) $\sigma = 0.2$



(b) $\sigma = 0.4$

Figure 25 10. SQNR vs. wordlength (radix-4, normally distributed input).

Figure 26 illustrates the SQNR outcomes for various standard deviations from 0.05 to 0.5. It is evident that the BFP-based method and ours are relatively insensitive to the changes of standard deviation. On the contrary, the value of standard deviation makes a tremendous impact on the performance of both the Oppenheim's and the Ramakrishnan's. As expected, the smaller the standard deviation is, the worse the performance of the Oppenheim's is. Since a smaller deviation suggests a lower occurrence probability of overflow, it should not be a good idea to increase one (two) bits of the integer part for every radix-2 (4) butterfly stage.

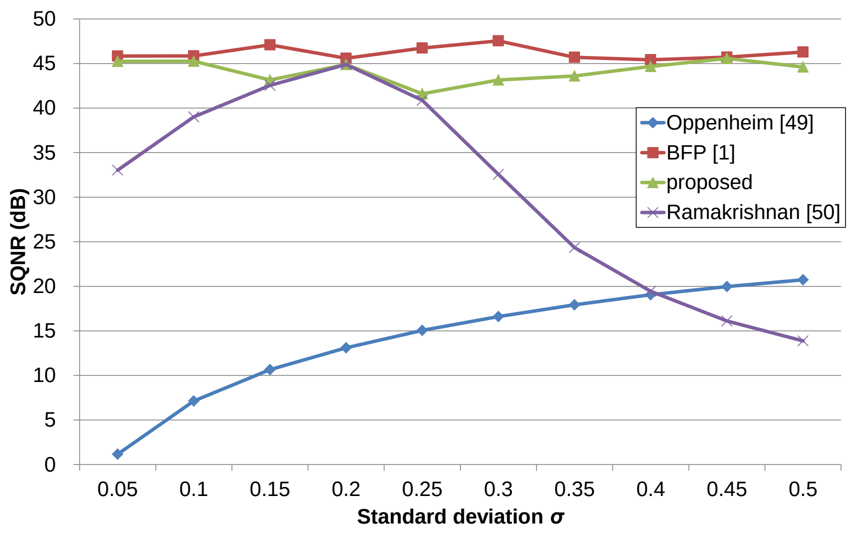


Figure 26 SQNR vs. standard deviation (radix-4, input in 12**b**1**f**).

4.5.3 SQNR for a real case study

In this experiment, we use a piece of 16-bit 11 KHz audio in WAV format from Wikipedia [80]. The PMF of the audio data is given in Figure 27, which is very close to a normal distribution with a mean of zero and a standard deviation of 0.168. Table 13 presents the number of integer bits of each stage and the scaling optimization outcomes for 256-point radix-2 FFT. Oppenheim's method [49], which increases the integer part by one bit for every single stage, gives a moderate result (SQNR=46.07dB). Ramakrishnan's method [50], which increases the integer part by

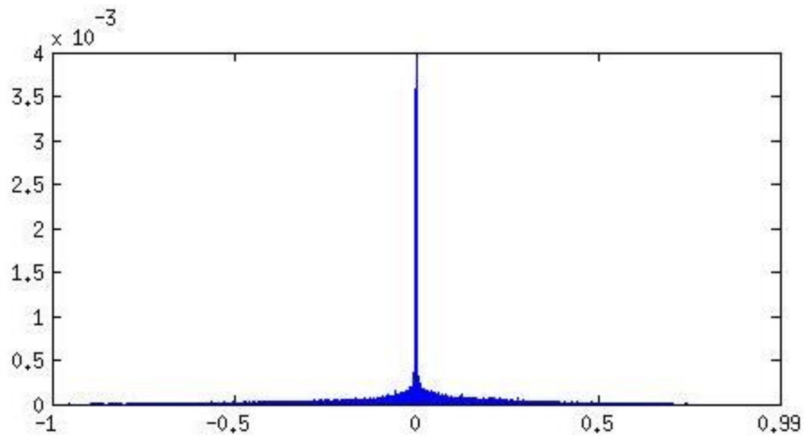


Figure 27 The PMF of the piece of music.

one bit for every two stages, suffers a serious overflow problem under this particular input and the resultant output SQNR is thus extremely low (10.18dB).

Since the additional scale factor tables in BFP determine an appropriate number format by detecting the largest value for each block, the 16-bit wordlength is long enough to preserve the high accuracy and significantly diminish the chance of overflow. Thus, the BFP method [1] achieves the best SQNR result. Meanwhile, if a uniformly distributed input is assumed in our proposed method (not the correct distribution), the output SQNR can achieve 51.75dB, which is still much higher than that of the Oppenheim's method [49]. Moreover, if the correct PMF of that piece of music (normal distribution with a mean of zero and a standard deviation of 0.168) is used in the analysis, the resultant output SQNR is even better (54.04dB). The experimental results here demonstrate that our proposed method can offer excellent quality of result if the PMF of the input is known in advance. Even if the PMF of the input is unknown, the quality of result is still fairly good in this experiment when compared with those two previous static scaling works.

Table 13 The Scaling optimization outcomes for 256-point radix-2 FFT

Method	Integer bits of each stage								SQNR
	1	2	3	4	5	6	7	8	
Oppenheim [49]	2	3	4	5	6	7	8	9	46.07dB
Ramakrishnan [50]	2	2	3	3	4	4	5	5	10.18dB
BFP [1]	N/A								85.67dB
Proposed (Uniform distribution)	2	3	4	5	6	6	7	7	51.75dB
Proposed (Normal distribution)	2	3	4	5	5	6	6	7	54.04dB

4.6 Summary

In this dissertation, we present an efficient static scaling technique for output SQNR optimization targeting FFT processor designs with the fixed wordlength constraint. Unlike most of conventional methods, it relies on fast probability-based analysis instead of time-consuming simulation to precisely model the noises induced from quantization and saturation. It works with various FFT sizes, radices, wordlengths, and most commonly presumed input distributions. The experimental results clearly show that the proposed technique is superior to existing static scaling approaches in all circumstances [49, 50]. Specifically, our technique can generate an 8192-point radix-2 FFT implementation with three bits shorter in wordlength than Oppenheim's method [49] while still achieving the same output SQNR, which implies a significant improvement in area, latency, and power consumption. Besides, it generally performs as nearly well as the BFP-based dynamic scaling technique [1], which requires an extra hardware unit. Therefore, we believe our fast static probability-based scaling optimization technique is very practical and helpful for creating area-efficient fixed-wordlength (e.g., memory-based) FFT processor designs

Chapter 5

Conclusion and Future Works

5.1 Conclusion

Since the DSP algorithms take an important role in the communication systems, the hardware implementations must carefully consider many parameters, such as the bitwidth, the base architecture, and the number scaling. In order to deal with the design issues at crafting the DSP algorithms, we propose three techniques in this dissertation: 1) a bitwidth-aware synthesis algorithm for MCM designs, 2) an EMDC-based FFT architecture, and 3) a static scaling technique for pipelined FFT processor designs.

First, we present an ILP-based bitwidth-aware area minimization algorithm for MCM designs, which points out that the total adder bit count rather than the total adder count can better estimate the hardware cost in a real implementation. Then, for a given MCM design, those target constants are first represented in a specified number format. Next, a subexpression graph is created to record all feasible decompositions for every target constant. The graph also keeps track of the required adder bitwidth as well as two subexpressions for every decomposition. At last, the area minimization problem is formulated as a set of ILP constraints derived from the subexpression graph and optimally resolved within an acceptable runtime.

Then, we propose an expandable multi-path delay commutator (EMDC) based FFT architecture. We show that the proposed architecture can be easily and flexibly expanded to satisfy throughput-hungry applications. In addition, a parameterizable hardware generator is also developed to automatically produce the specified HDL code so that the design cost and time can be drastically minimized. Finally, the theoretical analyses and/or experimental results demonstrate that the proposed architecture does consume less area and power than the existing foldable Pease architecture under the same throughput constraint.

Finally, we present an efficient static scaling technique for output SQNR optimization targeting FFT processor designs with the fixed wordlength constraint. Without using the time-consuming simulation to precisely model the induced quantization and saturation noises, we proposed a probability-based static scaling analysis to model the probabilistic behavior of the output signal at each stage. It works with various FFT sizes, radices, wordlengths, and most commonly presumed input distributions. The experimental results clearly show that the proposed technique is superior to existing static scaling approaches in all circumstances [49, 50]. Specifically, our technique can generate an 8K-point radix-2 memory-based FFT processor without compromise in the output SQNR. Besides, it generally performs as nearly well as the BFP-based dynamic scaling technique [1], which requires an additional hardware unit.

5.2 Future Works

Although the problem of multiplier-less constant multiplication has been studied for many decades, it still attracts a large number of attentions. In general, the digit-based algorithms can provide a better solution than the graph-based algorithms, but the quality of result of the digit-based algorithms is highly depended on the decomposition method. For example two numbers, 3 and 29, can be decomposed as follows in MSD form:

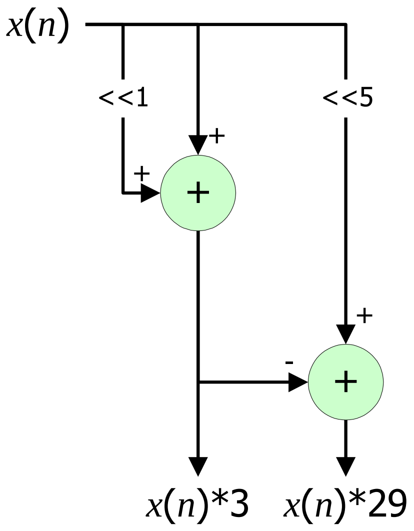


Figure 28 An alternative implementation for the number 3 and 29.

$$3_{10} = 11_2 = 10\bar{1}_{\text{MSD}} \text{ or } 011_{\text{MSD}}$$

and

$$29_{10} = 11101_2 = 100\bar{1}01_{\text{MSD}}$$

Thus, at least 3 adders are required in digit-based methods. However, a graph-based algorithm, RAG-n [14], allows the usage of right shifters and accepts a mapping result that induces extra adders for a coefficient to maximize the global subexpression sharing, it is capable of finding a design solution that is not presented in the digit-based algorithms. Figure 28 shows an alternative design that only two adders are needed. Obviously, the solution is outside of the current digit-based decomposition. Thus, the exploration of finding a valid and a limited number of decomposition is a research topic.

Furthermore, in order to achieve the high-data-rate communication, many new algorithms are proposed. For example, polor codes [81, 82] derived from the concept of channel polarization have emerged as the important channel codes for the capacity-achieving property. The hardware architecture of a (8, 4) polar code is given in Figure 29. Similar to the FFT core design, the same techniques on the hardware folding and the scaling analysis can be applied to the polar decoder. However, the data propagation is always performed in logarithmic domain which is not a linter time-invariant (LTI) operation. The non-linear property needs further research to apply the proposed techniques.

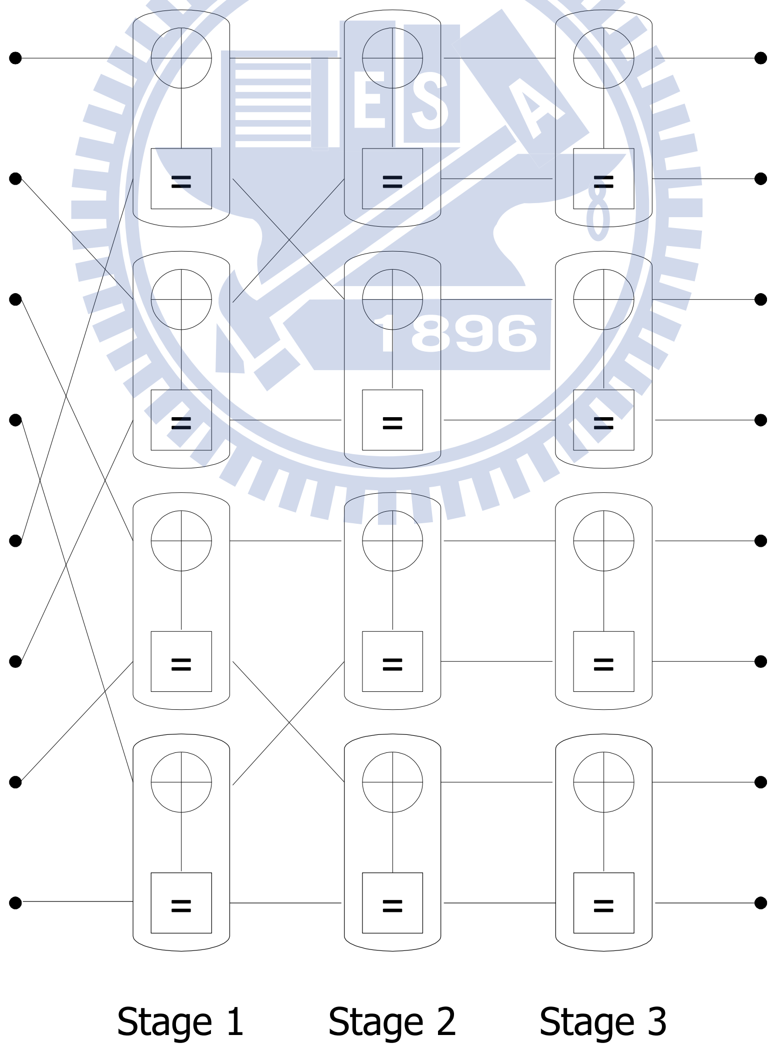


Figure 29 The factor graph of (8,4) polar code.

References

[1] Y.-W. Lin, H.-Y. Liu, and C.-Y. Lee, “A dynamic scaling FFT processor for DVB-T applications,” *IEEE Journal of Solid-State Circuits*, vol. 39, no. 11, pp. 2005–2013, Nov. 2004.

[2] C.-T. Lin, Y.-C. Yu, and L.-D. Van, “A low-power 64-point FFT/IFFT design for IEEE 802.11a WLAN application,” *IEEE International Symposium on Circuits and Systems*, 2006 (*ISCAS* 2006), pp. 4523–4526, May 2006.

[3] S. Li, H. Xu, W. Fan, Y. Chen, and X. Zeng, “A 128/256-point pipeline FFT/IFFT processor for MIMO OFDM system IEEE 802.16e,” *IEEE International Symposium on Circuits and Systems*, 2010 (*ISCAS* 2010), pp. 1488–1491, May 2010.

[4] Y. Chen, Y.-W. Lin, Y.-C. Tsao, and C.-Y. Lee, “A 2.4-Gsample/s DVFS FFT processor for MIMO OFDM communication systems,” *IEEE Journal of Solid-State Circuits*, vol. 43, no. 5, pp. 1260–1273, May 2008.

[5] C.-M. Chen, C.-C. Hung, and Y.-H. Huang, “An energy-efficient partial FFT processor for the OFDMA communication system,” *IEEE Transactions on Circuits and Systems II*, vol. 57, no. 2, pp. 136–140, Feb. 2010.

[6] S.-N. Tang, C.-H. Liao, and T.-Y. Chang, “An area- and energy-efficient multimode FFT processor for WPAN/WLAN/WMAN systems,” *IEEE Journal of Solid-State Circuits*, vol. 47, no. 6, pp. 1419–1435, Jun. 2012.

[7] Y.-W. Lin and C.-Y. Lee, “Design of an FFT/IFFT processor for MIMO OFDM systems,” *IEEE Transactions on Circuits and Systems I*, vol. 54, no. 4, pp. 807–815, Apr. 2007.

[8] M.-L. Ku and C.-C. Huang, “A complementary codes pilot-based transmitter diversity technique for OFDM systems,” *IEEE Transactions on Wireless Communications*, vol. 5, no. 3, pp. 504－504, Mar. 2006.

[9] E. Grass, K. Tittelbach-Helmrich, U. Jagdhold, A. Troya, G. Lippert, O. Kruger, J. Lehmann, K. Maharatna, K.F. Dombrowski, N. Fiebig, R. Kraemer, and P. Mahonen, “On the single-chip implementation of a Hiperlan/2 and IEEE 802.11a capable modem,” *IEEE Personal Communications*, vol. 8, no. 6, pp. 48–57, Dec. 2001.

[10] M. Krstić, K. Maharatna, A. Troya, E. Grass, and U. Jagdhold, “Baseband processor for IEEE 802.11a standard with embedded BIST,” Facta Universitatis, Series: Electronics and Energetics, pp. 231－239, Feb. 2007.

[11] P. Cappello and K. Steiglitz, “Some complexity issues in digital signal processing,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*,

vol. 32, no. 5, pp. 1037－1041, Oct. 1984.

[12] N. Sidahao, G.A. Constantinides, and P.Y. Cheung, “A heuristic approach for multiple restricted multiplication,” *IEEE International Symposium on Circuits and Systems*, 2005 (*ISCAS* 2005), pp. 692－695, May 2005.

[13] D. Bull and D. Horrocks, “Primitive operator digital filters,” *IEE Proceedings G on Circuits, Devices and Systems*, vol. 138, no. 3, pp. 401－412, Jun. 1991.

[14] A.G. Dempster and M.D. Macleod, “Use of minimum-adder multiplier blocks in FIR digital filters,” *IEEE Transactions on Circuits and Systems II*: *Analog and Digital Signal Processing*, vol. 42, no. 9, pp. 569－577, Sep. 1995.

[15] M. Mehendale, S.D. Sherlekar, and G. Venkatesh, “Synthesis of multiplier-less FIR filters with minimum number of additions,” *IEEE/ACM International Conference on Computer-Aided Design*, 1995 (*ICCAD* 1995), pp. 668－671, Nov. 1995.

[16] H.-J. Kang, H. Kim, and I.-C. Park, “FIR filter synthesis algorithms for minimizing the delay and the number of adders,” *IEEE/ACM International Conference on Computer Aided Design*, 2000 (*ICCAD* 2000), pp. 51－54, Nov. 2000.

[17] I.-C. Park and H.-J. Kang, “Digital filter synthesis based on minimal signed digit representation,” *Proceedings on Design Automation Conference*, 2001 (*DAC* 2001), pp. 468－473, Jun. 2001.

[18] M. Kumm, P. Zipf, M. Faust, and C.-H. Chang, “Pipelined adder graph optimization for high speed multiple constant multiplication,” *IEEE International Symposium on Circuits and Systems*, 2012 (*ISCAS* 2012), pp. 49－52, May 2012.

[19] Y. Voronenko and M. Püschel, “Multiplierless multiple constant multiplication,” *ACM Transactions on Algorithms* (*TALG*), vol. 3, no. 2, pp. 1549-6325, May 2007.

[20] R.I. Hartley, “Subexpression sharing in filters using canonic signed digit multipliers,” *IEEE Transactions on Circuits and Systems II*: *Analog and Digital Signal Processing*, vol. 43, no. 10, pp. 677－688, Oct. 1996.

[21] R. Pasko, P. Schaumont, V. Derudder, S. Vernalde, and D. Durackova, “A new algorithm for elimination of common subexpressions,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 1, pp. 58－68, Jan. 1999.

[22] R. Hewlitt and E. Swartzlantler, “Canonical signed digit representation for FIR digital filters,” *IEEE Workshop on Signal Processing Systems*, 2000 (*SiPS* 2000), pp. 416－426, 2000.

[23] O. Gustafsson and L. Wanhammar, “ILP modelling of the common

subexpression sharing problem," *International Conference on Electronics, Circuits and Systems*, 2002 (*ICECS* 2002), vol. 3, pp. 1171－1174, 2002.

[24] C.-Y. Yao, H.-H. Chen, T.-F. Lin, C.-J. Chien, and C.-T. Hsu, "A novel common-subexpression-elimination method for synthesizing fixed-point FIR filters," *IEEE Transactions on Circuits and Systems I*: *Regular Papers*, vol. 51, no. 11, pp. 2215－2221, Nov. 2004.

[25] Y. Wang and K. Roy, "CSDC: a new complexity reduction technique for multiplierless implementation of digital FIR filters," *IEEE Transactions on Circuits System I*: *Regular Papers*, vol. 52, no. 9, pp. 1845－1853, Sep. 2005.

[26] O. Gustafsson, "Lower Bounds for Constant Multiplication Problems," *IEEE Transactions on Circuits and Systems II*: *Express Briefs*, vol. 54, no. 11, pp. 974－978, Nov. 2007.

[27] L. Aksoy, E. da Costa, P. Flores, and J. Monteiro, "Exact and approximate algorithms for the optimization of area and delay in multiple constant multiplications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 6, pp. 1013－1026, Jun. 2008.

[28] Y.A. Ho, C. Lei, H. Kwan, and N. Wong, "Global optimization of common subexpressions for multiplierless synthesis of multiple constant multiplications," *IEEE Asia and South Pacific Design Automation Conference*, 2008 (*ASPDAC* 2008), pp. 119－124, Mar. 2008.

[29] Y.A. Ho, C. Lei, H. Kwan, and N. Wong, "Optimal common sub-expression elimination algorithm of multiple constant multiplications with a logic depth constraint," *IEICE Transactions on Fundamentals of Electronics, Communications, and Computer Science*, vol. E91-A, no. 12, pp. 3568－3575, Dec. 2008.

[30] L. Aksoy, E.O. Gunes, and P. Flores, "An exact breadth-first search algorithm for the multiple constant multiplications problem," *NORCHIP conference*, 2008, pp. 41－46, Nov. 2008.

[31] J. Thong and N. Nicolici, "Combined optimal and heuristic approaches for multiple constant multiplication," *IEEE International Conference on Computer Design*, 2010 (*ICCD* 2010), pp. 266－273, Oct. 2010.

[32] R. Mahesh and A. Vinod, "New reconfigurable architectures for implementing FIR filters with low complexity," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 2, pp. 275－288, Feb. 2010.

[33] J. Thong and N. Nicolici, "An Optimal and Practical Approach to Single Constant Multiplication," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 9, pp. 1373－1386, Sep. 2011.

[34] M. R. Garey and D. S. Johnson, Computers and intractability: a guide to the theory of NP-completeness, W. H. Freeman & Co., 1979.

[35] P. J. Downey, R. Sethi, and R. E. Tarjan, “Variations on the common subexpression problem,” *Journal of the ACM* (*JACM*), vol. 27, no. 4, pp. 758－771, Oct. 1980.

[36] J. W. Cooley and J. W. Turkey, “An algorithm for machine computation of complex Fourier series,” *Mathematics of Computation*, vol. 19, pp. 297–301, 1965.

[37] L. R. Rabiner and B. Gold, *Theory and application of digital signal processing*, Prentice-Hall, Inc., 1975.

[38] E. H. Wold and A. M. Despain, “Pipeline and parallel-pipeline FFT processors for VLSI implementations,” *IEEE Transactions on Computers*, vol. C-33, no. 5, pp. 414–426, May 1984.

[39] A. M. Despain. “Fourier transform computers using CORDIC iterations,” *IEEE Transactions on Computers*, vol. C-23, no. 10, pp. 993–1001, Oct. 1974.

[40] S. He and M. Torkelson, “A new approach to pipeline FFT processor,” *The 10th International Parallel Processing Symposium*, 1996 (*IPPS* 1996), pp. 766–770, Apr. 1996.

[41] R. Storn. “Radix-2 FFT-pipeline architecture with reduced noise-to-signal ratio,” *IEE Proceedings of Vision, Image, and Signal Processing*, vol. 141, no. 2, pp. 81–86, Apr. 1994.

[42] S. He and M. Torkelson, “Designing pipeline FFT processor for OFDM (de)modulation,” *International Symposium on Signals, Systems, and Electronics*, 1998 (*ISSSE* 1998) , pp. 257–262, Sep. 1998.

[43] C. Cheng and K. K. Parhi, “High-throughput VLSI architeture for FFT computation,” *IEEE transactions on Circuits and Systems II*, vol. 54, no. 10, pp. 863–867, Oct. 2007.

[44] A. Cortes, I. Velez, and J. Sevillano, “Radix r^k FFTs: matricial representation and SDC/SDF pipeline implementation,” *IEEE transactions on Signal Processing*, vol. 57, no. 7, pp. 2824–2839, Jul. 2009.

[45] S.-N. Tang, J.-W. Tsai, and T.-Y. Chang, “A 2.4-GS/s FFT processor for OFDM-based WPAN applications,” *IEEE Transactions on Circuits and Systems II*, vol. 57, no. 6, pp. 451–455, Jun. 2010.

[46] Y. Chen, Y.-C. Tsao, Y.-W. Lin, C.-H. Lin, and C.-Y. Lee, “An indexed-scaling pipelined FFT processor for OFDM-based WPAN applications,” *IEEE Transactions on Circuits and Systems II,* vol. 55, no. 2, pp. 146–150, Feb. 2008.

[47] W.-H. Chang and T.Q. Nguyen, “On the fixed-point accuracy analysis of FFT

algorithms," *IEEE Transactions on Signal Processing*, vol. 56, no. 10, pp. 4673–4682, Oct. 2008.

[48] C.-Y. Wang, C.-B. Kuo, and J.-Y. Jou, "Hybrid wordlength optimization methods of pipelined FFT processors," *IEEE Transactions on Computers*, vol. 56, no. 8, pp. 1105–1118, Aug. 2007.

[49] A.V. Oppenheim and C.J. Weinstein, "Effects of finite register length in digital filtering and the fast Fourier transform," *Proceedings of the IEEE*, vol. 60, no. 8, pp. 957–976, Aug. 1972.

[50] S. Ramakrishnan, J. Balakrishnan, and K. Ramasubramanian, "Exploiting signal and noise statistics for fixed point FFT design optimization in OFDM systems," *2010 National Conference on Communications (NCC)*, pp. 1–5, Jan. 2010.

[51] E. Ya. Remez, General computational methods of Chebyshev approximation: The problems with linear real parameters, Washington, DC: Atomic Energy Commission Translation Series, 1957.

[52] T. Parks and J. McClellan, "Chebyshev Approximation for Nonrecursive Digital Filters with Linear Phase," *IEEE Transactions on Circuit Theory*, vol. 19, no. 2, pp. 189－194, Mar. 1972.

[53] Gurobi Optimizer. [Online]. Available: http://www.gurobi.com

[54] G. Nordin, P. A. Milder, J. C. Hoe, and M. Püschel, "Automatic generation of customized discrete Fourier transform IPs," *ACM/IEEE Design Automation Conference*, 2005 (*DAC* 2005) , pp. 471–474, Jun. 2005.

[55] P. A. Milder, F. Franchetti, J. C. Hoe, and M. Püschel, "Formal datapath representation and manipulation for implementing DSP transforms," *ACM/IEEE Design Automation Conference*, 2008 (*DAC* 2008), pp. 385–390, Jun. 2008.

[56] E. Bidet, D. Castelain, C. Joanblanq, and P. Senn, "A fast single-chip implementation of 8192 complex point FFT," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 3, pp. 300–305, Mar. 1995.

[57] T. Lenart and V. Owall, "A 2048 complex point FFT processor using a novel data scaling approach," *IEEE International Symposium on Circuits and Systems*, 2003 (*ISCAS* 2003), pp. IV-45–IV-48, May 2003.

[58] A. Mallik, D. Sinha, P. Banerjee, and H. Zhou, "Low-power optimization by smart bit-width allocation in a SystemC-based ASIC design environment," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 26, no. 3, pp. 447–455, Mar. 2007.

[59] M. Haldar, A. Nayak, N. Shenoy, A. Choudhary, and P. Banerjee, "FPGA hardware synthesis from MATLAB," *International Conference on VLSI*

Design, 2001, pp. 299–304, Jan. 2001.

[60] G. A. Constantinides, P. Y. K. Cheung, and W. Luk, "The multiple wordlength paradigm," *The 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2001, pp. 51–60, Mar. 2001.

[61] O. Sarbishei and K. Radecka, "Analysis of mean-square-error (MSE) for fixed-point FFT units," *IEEE International Symposium on Circuits and Systems*, 2011 (*ISCAS* 2011), pp. 1732–1735, May 2011.

[62] W.-H. Chang and T. Q. Nguyen, "On the fixed-point accuracy analysis of FFT algorithms," *IEEE Transactions on Signal Processing*, vol. 56, no. 10, pp. 4673–4682, Oct. 2008.

[63] G. Caffarena, C. Carreras, J. A. López, and Á. Fernández, "SQNR estimation of fixed-point DSP algorithms," *EURASIP Journal on Advances in Signal Processing*, vol. 2010, no. 21, pp. 1–12, Feb. 2010.

[64] G. A. Constantinides, P. Y. K. Cheung, and W. Luk, "Wordlength optimization for linear digital signal processing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 10, pp. 1432–1442, Oct. 2003.

[65] R. Rocher, D. Menard, O. Sentieys, and P. Scalart, "Analytical accuracy evaluation of fixed-point systems," *15th European Signal Processing Conference*, 2007 (*EUSIPCO07*), pp. 999–1003, Sep. 2007.

[66] D. Menard, R. Serizel, R. Rocher, and O. Sentieys, "Accuracy constraint determination in fixed-point system design," *EURASIP Journal on Embedded Systems*, vol. 2008, no. 1, pp. 1–12, Oct. 2008.

[67] D. Menard, R. Rocher, and O. Sentieys, "Analytical Fixed-Point Accuracy Evaluation in Linear Time-Invariant Systems," *IEEE Transactions on Circuits and Systems I*, vol. 55, no. 10, pp. 3197–3208, Nov. 2008.

[68] O. Sarbishei and K. Radecka, "On the fixed-point accuracy analysis and optimization of FFT units with DORDIC multipliers," *IEEE Symposium on Computer Arithmetic*, 2011 (*ARITH*), pp. 62–69, Jul. 2011.

[69] H. Keding, M. Willems, M. Coors, and H. Meyr, "FRIDGE: a fixed-point design and simulation environment," *Proceedings of Design, Automation and Test in Europe*, 1998, pp. 429–435, Feb. 1998.

[70] D.-U. Lee, A. A. Gaffar, R. C. C. Cheung, O. Mencer, W. Luk, and G. A. Constantinides, "Accuracy-guaranteed bit-width optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 1990–2000, Oct. 2006.

[71] J.W. Cooley and J.W. Tukey, "An algorithm for machine computation of complex Fourier series," *Math. Computation*, vol. 19, pp. 45–48, 1965.

[72] W.-C. Yeh and C.-W. Jen, “High-speed and low-power split-radix FFT,” *IEEE Transactions on Signal Processing*, vol. 51, no. 3, pp. 864–874, Mar. 2003.

[73] Y.-W. Lin, H.-Y. Liu, and C.-Y. Lee, “A 1-GS/s FFT/IFFT processor for UWB applications,” *IEEE Journal of Solid-State Circuits,* vol. 40, no. 8, pp. 1726–1735, Aug. 2005.

[74] R.C. Agarwal and J.W. Cooley, “Vectorized mixed radix discrete Fourier transform algorithms,” *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1283–1292, Sep. 1987.

[75] A.V. Oppenheim and R.W. Schafer, *Discrete-time signal processing*, Prentice Hall, Englewood Cliffs, NJ, 2009.

[76] D.P. Bertsekas and J.N. Tsitsiklis, *Introduction to probability*, Athena Scientific, Nashua, NH, 2002.

[77] C. M. Grinstead and J. L. Snell, *Introduction to probability*, American Mathematical Society, Providence, RI, 1997.

[78] T.-C. Wei, W.-C. Liu, and S.-J. Jou, “A jointed mode detection and symbol detection scheme for DVB-T,” *IEEE Transactions on Consumer Electronics*, vol. 54, no. 2, pp. 336–341, May 2008.

[79] L. Yun, J. Zhou, and Y. Onozato, “Performance study for STC-OFDM systems based on IEEE802.11a standard,” *International Conference on Wireless Communications, Networking, and Mobile Computing*, 2009 (*WiCom ‘09*), pp. 1–4, Sep. 2009.

[80] Wikipedia: Waveform Audio File Format. [Online] Available: http://en.wikipedia.org/wiki/WAV.

[81] B. Y. and K. K. Parhi, “Architecture optimizations for BP polar decoders,” *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013 (*ICASSP*), pp. 2654–2658, May 2013.

[82] B. Yuan and K. K. Parhi, “Low-latency successive-cancellation polar decoder architectures using 2-bit decoding,” *IEEE Transactions on Circuits and Systems I*, vol. 61, no. 4, pp. 1241–1254, Apr. 2014.